МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МИКИТЕНКО ВЕРОНІКА ЮРІЇВНА

Допускається до захисту:
завідувач кафедри інформаційних технологій, д. т. н., доцент
_____ Т. В. Нескородєва
«_____» _____ 2022р.

РОЗРОБКА ВЕБ-ДОДАТКУ ПОШУКУ РОБОТИ З ІНТЕЛЕКТУАЛЬНОЮ СИСТЕМОЮ ПІДБОРУ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота
(відповідно до стандарту спеціальності та ОП)

Науковий керівник:
Т. В. Січко, доцент кафедри інформаційних технологій
к. т. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця 2022

АНОТАЦІЯ

Микитенко В.Ю. Розробка веб-додатку пошуку роботи з інтелектуальною системою підбору. Спеціальність 122 «Комп'ютерні науки», Освітня програма «Комп'ютерні технології обробки даних». Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У кваліфікаційній (магістерській) роботі було розроблено та імплементовано архітектуру веб-додатку з реалізацією технологій Application Programming Interface, описано створені ендпоїнти, досліджено та реалізовано алгоритми машинного навчання на основі моделі лінійної регресії, створено допоміжні скрипти для полегшення тестування та деплойменту.

Ключові слова: API, Application Programming Interface, Python, машинне навчання, дані, CRUD, модель, тестування.

88 с., 2 табл., 17 рис., 25 джерел.

Mykytenko V. Y. Development of work searching web application with an intelligent selection system. Specialty 122 "Computer science", Programme "Computer data processing technologies". Vasyl` Stus Donetsk National University, Vinnytsia, 2022.

As a result of the qualification (master's) thesis was developed web service with microservice architecture and implementation of the Application Programming Interface mechanism. All of the endpoints were described in the SWAGGER documentation page with ability of sending test requests to listed endpoints. Under the investigation was also created machine learning algorithm for selecting of most compatible resume and vacancy with the usage of Linear Regression model. Repository also contain utilities script for easier testing and deployment process.

Keywords: API, Application Programming Interface, Python, Machine Learning, data, CRUD, model, testing

88 pages, 2 tabl., 25 sources

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АЗ – Апаратне забезпечення

БД – база даних

ПЗ – програмне забезпечення

API – Application Programming Interface

JSON – Javascript Object Notation

REST – Representational State Transfer

SQL – Structured Query Language

Деплоймент – процес розгортання програмного продукту у готовність до використання; усі дії, що роблять програмну систему готовою до використання.

Route (маршрут) – URL, до якого можна звернутись різними HTTP методами. Може мати декілька ендпоїнтів.

Ендпоїнт – саме звернення до маршруту окремим HTTP методом. Виконує конкретну задачу, приймає параметри та повертає дані Клієнту.

Імплементація – програмна чи апаратна реалізація якого-небудь протоколу, алгоритму чи технології.

Workflow – робочий процес.

ВСТУП

В сучасному світі інформація є одним із найважливіших ресурсів людства. Для коректної роботи з нею, обробки та аналізу було винайдено багато інструментів, які з часом вдосконалювалися та розширялися новим функціоналом. Було здійснено перехід від розробки додатків як монолітів – єдиних алгоритмів, які включають в себе всі можливі функції утиліти, - до мікросервісної архітектури додатків, що дозволяє швидше та якісніше оновлювати той чи інший функціонал, відокремлювати маніпулювання різними даними один від одного. Найкращим інструментом для побудови мікросервісів на даний момент вважають реалізацію технології API.

Фактично, механізм Application Programming Interface допомагає створювати різні реалізації одного й того ж алгоритму для різних платформ та продуктів. API зазвичай є найбільш вдалим способом реалізувати бекенд веб-додатку, який у результаті може біти представлений як у версії сайту з фронтендом, так і в десктоп додатку, або мобільному додатку для платформ Android та IOS. Буде відрізнятися механізм імплементації апаратної частини, проте всі процеси бізнес логіки та взаємодії з БД будуть проходити на одному сервері, що в подальшому дозволить зберегти консистентність даних.

Таким чином, автором було поставлено за мету розробити універсальний веб-додаток з реалізацією механізму Application Programming Interface для маніпулювання даними резюме та вакансій на основі мови програмування Python та з використанням асинхронного веб-фреймворку FastAPI.

Виходячи із мети роботи, об’єктом дослідження є процес розробки та реалізації технологій API та веб-додатків, а предметом – реалізація Application Programming Interface з використанням технологій machine learning.

Аналізуючи поставлену мету, автором було виділено наступні задачі, які мають бути реалізовані в ході виконання роботи:

- розробити основний сервер веб-API та створити ендпоїнти для маніпуляції даними користувачів, компаній, резюме та вакансій;

- створити архітектуру бази даних, яка буде відповідати за зберігання даних та проміжних моделей веб-додатку;
- реалізувати механізм авторизації та аутентифікації запитів до серверу веб-API;
- створити сторінку документації ендпоїнтів з можливістю їх тестування;
- імплементувати покриття коду тестами не менше ніж на 80% функціоналу;
- розробити додаткові скрипти та докер-файли для спрощення деплойменту додатку на сервер та його тестування;
- реалізувати алгоритми машинного навчання для підбору найкращих резюме до вакансій та навпаки.

Обрана тема є актуальною, оскільки на сьогоднішній день реалізація мікросервісної архітектури є ключовим напрямком розробки на мові програмування Python, а сайти для пошуку роботи саме в IT-сфері затребуваними. Наукова новизна роботи полягає в реалізації алгоритмів автоматичного підбору найліпшого співпадіння вакансії та резюме, що має значно полегшити роботу HR-працівників та рекрутерів у пошуку нових кандидатів на співбесіди. Цим також зумовлена практична цінність проведеного дослідження.

Результати кваліфікаційної (магістерської) роботи апробовано на III Всеукраїнській науково-практичній конференції «Комп’ютерні технології обробки даних», яка відбулась 8 грудня 2022 року на базі кафедри інформаційних технологій Донецького національного університету імені Василя Стуса. Тези на тему: «Проектування додатку пошуку роботи з вбудованою системою підбору» були опубліковані в електронному збірнику наукових праць. Також результати викладені на код-ресурс GitHub під OpenSource ліцензією для можливого подальшого використання іншими користувачами.

Кваліфікаційна робота складається з чотирьох розділів, висновків та списку використаних джерел.

У першому розділі розглянуто теоретичну базу дослідження та проведено порівняльний аналіз можливих засобів реалізації поставлених задач.

У другому розділі створено актуальну модель майбутнього бекенду веб-додатку, розроблено основні алгоритми машинного навчання, прописано основні структури даних, визначені та сформульовані вимоги до функціоналу Application Programming Interface. Також побудовано логічну модель розроблюваного застосунку та визначено основні елементи фізичної моделі.

У третьому розділі визначено основні вимоги до бази даних з боку користувачів, виділено основні та допоміжні сутності предметної області. Побудовано діаграму потоків даних без декомпозиції та ЕР-діаграму – модель даних. Виявлено можливі обмеження зв’язків та полів бази даних, описано набори відношень між сутностями.

У четвертому розділі описано типові формати вхідних даних та модулі розробленого додатку, архітектуру чотирьох CRUD-класів, Session- та Router-класів. Виділено опис реалізації алгоритмів машинного. Розроблено інструкції для користувача. Описано документацію, розроблено автоматичні юніт- та інтеграційні тести. Наведено опис результатів Perfomance та End-to-End тестування.

Магістерська робота містить 88 сторінок, 2 таблиці, 17 рисунків та список літератури із 25 джерел.

РОЗДІЛ 1
ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ І ОГЛЯД ПІДХОДІВ ДО ВИРІШЕННЯ ЗАДАЧ

1.1 API та його різновиди як універсальний інструмент створення бекенду веб-додатків

1.1.1 Поняття Application Programming Interface

API — це механізми, які дозволяють двом програмним компонентам спілкуватися один з одним за допомогою набору визначень і протоколів. Наприклад, система програмного забезпечення бюро погоди містить щоденні дані про погоду. Програма погоди на телефоні «спілкується» з цією системою через API і показує щоденні оновлення погоди на телефоні [1].

API означає Application Programming Interface. У контексті API слово «Interface» стосується будь-якого програмного забезпечення з окремою функцією. Інтерфейс можна розглядати як договір на обслуговування між двома програмами. Цей контракт визначає, як вони спілкуються один з одним за допомогою запитів і відповідей. Їх документація API містить інформацію про те, як розробники структурують ці запити та відповіді.

API дозволяють продукту чи службі взаємодіяти з іншими продуктами та послугами, не знаючи, як вони реалізовані. Це може спростити розробку програми, заощадивши час і гроші. Коли розробляють нові інструменти та продукти або керують наявними, API дають гнучкість; спростити дизайн, адміністрування та використання; і надають можливості для інновацій.

API іноді розглядають як договори з документацією, яка представляє угоду між сторонами: якщо сторона 1 надсилає віддалений запит, структурований певним чином, саме так відповідатиме програмне забезпечення сторони 2.

API, що спрощує інтегрування нових компонентів додатків в уже існуючу архітектуру, допомагає встановленню комфортної та продуктивної взаємодії стек-холдерів (product owner-ів) та команд розробників.. Потреби бізнесу часто швидко змінюються у відповідь на постійні зміни цифрових ринків, де нові

конкуренти можуть змінити цілу галузь за допомогою нового додатка. Щоб залишатися конкурентоспроможними, важливо підтримувати швидкий розвиток і розгортання інноваційних послуг. Розробка додатків у хмарі — це помітний спосіб збільшити швидкість розробки, і він спирається на підключення архітектури додатків мікросервісів через API.

API — це спрощений спосіб підключення власної інфраструктури за допомогою розробки додатків у хмарі, але вони також дозволяють ділитися своїми даними з клієнтами та іншими зовнішніми користувачами. Загальнодоступні API представляють унікальну цінність для бізнесу, оскільки вони можуть спростити та розширити зв'язок із партнерами, а також потенційно монетизувати дані (популярним прикладом є API Карт Google).

1.1.2 Різновиди API в залежності від методу створення

Архітектура API зазвичай пояснюється з точки зору клієнта та сервера. Програма, яка надсилає запит, називається клієнтом, а програма, яка надсилає відповідь, називається сервером. Отже, у прикладі погоди база даних бюро погоди є сервером, а мобільний додаток є клієнтом [1].

Існують чотири різні способи роботи API залежно від того, коли та чому вони були створені:

- *SOAP API*. Ці API використовують простий протокол доступу до об'єктів. Клієнт і сервер обмінюються повідомленнями за допомогою XML. Це менш гнучкий API, який був більш популярним у минулому.
- *RPC API*. Ці API називаються віддаленими викликами процедур (Remote Procedure Calls). Клієнт виконує функцію (або процедуру) на сервері, а сервер надсилає результат назад клієнту.
- *Websocket API*. Ще одна сучасна розробка веб-API, яка використовує об'єкти JSON для передачі даних. WebSocket API підтримує двосторонній зв'язок між клієнтськими програмами та сервером. Сервер може надсилати повідомлення зворотного виклику підключеним клієнтам, що робить його ефективнішим, ніж REST API.

- *REST API*. Це найпопулярніші та найгнучкіші API, які сьогодні є в Інтернеті. Клієнт надсилає запити на сервер як дані. Сервер використовує цей вхід клієнта для запуску внутрішніх функцій і повертає вихідні дані назад клієнту. Нижче розглянемо REST API більш детально.

API класифікуються відповідно до їх архітектури та сфери використання. Основні типи архітектур API вже були представлені, тому подивимося на сферу використання.

1. Приватні API. Вони є внутрішніми для підприємства та використовуються лише для підключення систем і даних усередині підприємства.
2. Публічні API. Вони відкриті для громадськості та можуть використовуватися будь-ким. З цими типами API може виникати певна авторизація та вартість.
3. Партнерські API. Вони доступні лише уповноваженим зовнішнім розробникам для підтримки партнерства між компаніями.
4. Композитні API. Вони поєднують два або більше різних API для вирішення складних системних вимог або поведінки.

1.1.3 REST API та його переваги і недоліки

Що таке REST API

REST означає Representational State Transfer. REST визначає набір функцій, таких як GET, PUT, POST, DELETE, які клієнти можуть використовувати для доступу до даних сервера. Клієнти та сервери обмінюються даними за допомогою HTTP [2].

REST API використовує команди для отримання ресурсів. REST API використовує існуючі методології HTTP, визначені протоколом RFC 2616, наприклад:

1. GET для отримання ресурсу;

2. PUT, щоб змінити стан або оновити ресурс, який може бути об’єктом, файлом або блоком;
3. POST, щоб створити цей ресурс;
4. DELETE, щоб видалити ресурс.

Формати даних, які підтримує REST API, включають:

1. application/json
2. application/xml
3. application/x-wbe+xml
4. application/x-www-form-urlencoded
5. multipart/form-data

Головною особливістю REST API є відсутність стану. Відсутність стану означає, що сервери не зберігають клієнтські дані між запитами. Запити клієнтів до сервера схожі на URL-адреси, які вводяться у браузері, щоб відвідати веб-сайт. Відповідь від сервера є простими даними без типового графічного відтворення веб-сторінки [3].

API є REST, якщо вони відповідають 6 основним обмеженням системи REST:

1. *Архітектура клієнт-сервер.* Архітектура REST складається з клієнтів, серверів і ресурсів і обробляє запити через HTTP. Повинно бути чітке розмежування між клієнтом і сервером. Питання щодо інтерфейсу користувача та збору запитів належать до домену клієнта. Доступ до даних, керування навантаженням і безпека є доменом сервера. Цей слабкий зв’язок між клієнтом і сервером дозволяє розробляти та покращувати кожен незалежно від іншого.
2. *Відсутність стану.* Клієнтський вміст не зберігається на сервері між запитами. Натомість інформація про стан сеансу і будь-яке необхідне керування ним здійснюється на клієнті, а не на сервері.
3. *Кешування.* Усі ресурси мають дозволяти кешування, якщо не вказано, що кешування неможливе. Кешування може усунути потребу у деяких взаємодіях клієнт-сервер.

4. *Багаторівнева система.* REST дозволяє створити архітектуру, що складається з кількох рівнів серверів. Взаємодія клієнт-сервер може здійснюватися за допомогою додаткових рівнів. Ці рівні можуть пропонувати додаткові функції, такі як балансування навантаження, спільні кеші або безпека.

5. *Код на вимогу (необов'язково).* У більшості випадків сервер надсилає назад статичні представлення ресурсів у формі XML або JSON. Однак, коли це необхідно, сервери можуть надіслати виконуваний код клієнту. Сервери можуть розширювати функціональні можливості клієнта, передаючи виконуваний код.

6. *Уніфікований інтерфейс.* Це обмеження є основою дизайну REST API і включає 4 аспекти:

    1. *Ідентифікація ресурсів у запитах.* Ресурси мають бути однозначно ідентифіковані за допомогою однієї URL-адреси. Ресурси ідентифікуються в запитах і є окремими від представлень, що повертаються клієнту.

    2. *Маніпулювання ресурсами через представлення.* Тільки за допомогою основних методів мережевого протоколу, таких як DELETE, PUT і GET з HTTP, можна маніпулювати ресурсом Клієнти отримують файли, які представляють ресурси. Ці представлення повинні містити достатньо інформації, щоб дозволити зміну або видалення.

    3. *Повідомлення з описом.* Кожне повідомлення, яке повертається клієнту, містить достатньо інформації, щоб описати, як клієнт повинен обробляти інформацію.

    4. *Гіпермедіа як механізм стану програми.* Після доступу до ресурсу REST-клієнт повинен мати можливість виявляти за допомогою гіперпосилань усі інші доступні дії.

Що таке ресурс

Ключовою абстракцією інформації в REST є ресурс. Будь-яка інформація, яку можна назвати, може бути ресурсом. Наприклад, ресурсом REST може бути документ або зображення, тимчасова служба, набір інших ресурсів або невіртуальний об’єкт (наприклад, людиною) [2].

Стан ресурсу в будь-який конкретний момент часу називається представленням ресурсу.

Представлення ресурсів складаються з:

- дані;
- метадані, що описують дані;
- і гіпермедійні посилання, які можуть допомогти клієнтам у переході до наступного бажаного стану.

REST API складається з набору взаємопов’язаних ресурсів. Цей набір ресурсів відомий як модель ресурсів REST API.

REST використовує ідентифікатори ресурсів для ідентифікації кожного ресурсу, який бере участь у взаємодії між клієнтським і серверним компонентами.

REST API виглядає як гіпертекст. Кожна адресована одиниця інформації несе адресу, або явно (наприклад, атрибути посилання та id), або неявно.

Гіпертекст (або гіпермедіа) означає одночасне представлення інформації та засобів керування таким чином, що інформація стає можливістю, за допомогою якої користувач отримує вибір і вибирає дії.

Кожен тип носія визначає модель обробки за замовчуванням. Наприклад, HTML визначає процес відтворення гіпертексту та поведінку браузера навколо кожного елемента.

Що таке веб-API

Веб-API або API веб-сервісу — це інтерфейс обробки програми між веб-сервером і веб-браузером. Усі веб-служби є API, але не всі API є веб-службами. REST API — це особливий тип веб-API, який використовує стандартний архітектурний стиль, описаний вище.

Різні терміни щодо API, як-от Java API або API служби, існують, оскільки історично API були створені до появи всесвітньої мережі. Сучасні веб-API є REST API, і ці терміни можна використовувати як взаємозамінні.

Що таке API інтеграції

Інтеграції API — це програмні компоненти, які автоматично оновлюють дані між клієнтами та серверами. Деякими прикладами API інтеграції є автоматична синхронізація даних у хмару з галереї зображень телефону або автоматична синхронізація часу й дати на ноутбуці, під час подорожей в інший часовий пояс. Підприємства також можуть використовувати їх для ефективної автоматизації багатьох системних функцій.

Переваги REST API

REST API мають чотири основні переваги:

1. Інтеграція. API використовуються для інтеграції нових програм із існуючими програмними системами. Це збільшує швидкість розробки, оскільки кожну функціональність не потрібно писати з нуля. Можна використовувати API, щоб використовувати існуючий код.

2. Інноваційність. З появою нового додатка можуть змінитися цілі галузі. Підприємства повинні швидко реагувати та підтримувати швидке впровадження інноваційних послуг. Вони можуть зробити це, вносячи зміни на рівні API без необхідності переписувати весь код.

3. Розширення. API надають компаніям унікальну можливість задовольнити потреби своїх клієнтів на різних платформах. Наприклад, API карт дозволяє інтегрувати інформацію з карт через веб-сайти, Android, iOS тощо. Будь-яка компанія може надати аналогічний доступ до своїх внутрішніх баз даних, використовуючи безкоштовні або платні API.

4. Простота обслуговування. API діє як шлюз між двома системами. Кожна система зобов’язана вносити внутрішні зміни, щоб це не вплинуло на API. Таким чином, будь-які майбутні зміни коду однією стороною не впливатимуть на іншу сторону.

Загальні проблеми REST API

Крім обмежень дизайну та архітектури, доводиться стикатись з деякими проблемами REST API. Деякі поняття, які можуть бути складними, включають:

1. Узгодженість кінцевих точок — шляхи кінцевих точок мають бути узгодженими відповідно до загальних веб-стандартів, якими може бути важко керувати.
2. Керування версіями API — URL-адреси кінцевих точок не повинні вважатися недійсними під час використання внутрішньо або з іншими програмами.
3. Тривалий час відповіді та забагато даних — обсяг повернених ресурсів може з часом збільшуватися, що збільшує навантаження та час відповіді.
4. Шляхи навігації та місця введення користувачами — оскільки REST використовує URL-шляхи для вхідних параметрів, визначення URL-просторів може бути складним завданням.
5. Безпека – має багато аспектів, на які слід стежити, зокрема використання:
   - HTTPS;
   - блокування доступу з невідомих IP-адрес і доменів;
   - перевірка URL-адрес;
   - блокування несподівано великих корисних навантажень;
   - запити на реєстрацію;
   - розслідування невдач.
6. Автентифікація — використовує загальні методи автентифікації, такі як базова автентифікація HTTP, ключі API, веб-токени JSON та інші маркери доступу.
7. Запити та дані – запити можуть містити більше даних і метаданих, ніж потрібно, або для отримання всіх даних може знадобитися більше запитів. Для цього можна налаштувати API.
8. Тестування API – процес налаштування та запуску може бути тривалим. Кожна частина процесу може бути довгою або складною.

9. Визначення кодів помилок і повідомлень. Для кодів помилок більш поширеною практикою є використання стандартних кодів помилок HTTP. Обробка помилок може не мати способу визначити успішність відповіді чи ні, окрім аналізу тіла чи перевірки на наявність помилки.

1.1.4 SOAP API та його переваги і недоліки

SOAP — це простий протокол доступу до об'єктів, стандарт обміну повідомленнями, визначений World Wide Web Consortium та його членами-редакторами. SOAP використовує формат даних XML для оголошення своїх повідомлень запитів і відповідей, покладаючись на схему XML та інші технології для забезпечення структури своїх корисних навантажень.

Як публічні, так і приватні API використовують SOAP як інтерфейс. Організації будь-якого розміру виробляють і використовують SOAP API, хоча вони більш популярні на великих підприємствах.

SOAP використовує шаблон Remote Procedure Call (RPC), де функціям або методам передаються параметри та повертається результат. Багато рішень RPC до SOAP залежали від певних мов програмування або стеків технологій. Наприклад, попередні реалізації RPC часто вимагали, щоб обидві сторони RPC використовували мову програмування C, яка передувала сучасному Інтернету.

Серед важливих аспектів API SOAP є їхня незалежність від мови програмування та навіть базового транспортного протоколу. Наприклад, відправник може використовувати C#, тоді як стек одержувача покладається на Java. Хоча ці більш орієнтовані на підприємства мови найчастіше зустрічаються в SOAP, існують реалізації SOAP у Python, Ruby та всіх сучасних мовах програмування.

Останньою перевагою є то, що SOAP дуже розширюваний, але використовуються лише ті частини, які потрібні для конкретного завдання.

Складність залежить від мови програмування XML, який використовується для надсилання запитів і отримання відповідей у SOAP, може стати надзвичайно складним. У деяких мовах програмування потрібно

створювати ці запити вручну, що стає проблематичним, оскільки SOAP не терпить помилок. Однак інші мови можуть використовувати ярлики, які надає SOAP. Вони можуть допомогти зменшити зусилля, необхідні для створення запиту та аналізу відповіді.

Вбудована обробка помилок

Однією з найважливіших функцій SOAP є вбудована обробка помилок. Якщо з запитом виникла проблема, відповідь містить інформацію про помилку, яку можна використати для вирішення проблеми. Ця особливість є надзвичайно важливою, інакше доведеться здогадуватися, чому щось не працює. Звіти про помилки навіть надають стандартизовані коди, щоб можна було автоматизувати деякі завдання обробки помилок у коді.

Цікавою особливістю SOAP є те, що не обов'язково використовувати його з HTTP. Існує фактична специфікація для використання SOAP через простий протокол передачі пошти (SMTP), і немає жодних причин, чому не можна використовувати його в інших транспортних засобах. Насправді розробники деяких мов, таких як Python і PHP, роблять саме це [4].

1.1.5 Відмінності та методологія використання REST та SOAP API

REST і SOAP пропонують різні методи виклику веб-служби. REST — це архітектурний стиль, тоді як SOAP визначає стандартну специфікацію протоколу зв'язку для обміну повідомленнями на основі XML. Програми REST можуть використовувати SOAP.

Веб-служби REST не мають стану. Реалізація на основі REST проста порівняно з SOAP, але користувачі повинні розуміти контекст і вміст, що передається, оскільки немає стандартного набору правил для опису інтерфейсу веб-служб REST. Служби REST корисні для пристроїв з обмеженим профілем, наприклад мобільних, і їх легко інтегрувати з існуючими веб-сайтами.

SOAP вимагає менше низькорівневого інфраструктурного коду, який з'єднує головні модулі коду разом, ніж дизайн служб REST. Мова опису веб-служб описує загальний набір правил для визначення повідомлень, прив'язок,

операцій і розташування служби. Веб-служби SOAP корисні для асинхронної обробки та виклику [5].

SOAP має такі переваги порівняно з REST:

- Незалежність від мови, платформи та транспорту (REST вимагає використання HTTP)
- Добре працює в розподілених корпоративних середовищах (REST передбачає прямий зв'язок «крапка-крапка»)
- Стандартизований
- Забезпечує значні можливості розширення перед збіркою у формі стандартів WS
- Вбудована обробка помилок
- Автоматизація при використанні з певними мовними продуктами

REST здебільшого простіший у використанні та більш гнучкий. Він має такі переваги перед SOAP:

- Для взаємодії з веб-службою не потрібні дорогі інструменти
- Менша крива навчання
- Ефективність (SOAP використовує XML для всіх повідомлень, REST може використовувати менші формати повідомлень)
- Швидкий (не вимагає великої обробки)
- Ближче до інших веб-технологій у філософії дизайну

1.2 Python як мова програмування для реалізації додатку

1.2.1 Основні властивості та особливості мови програмування Python

Python — це автономна високорівнева мова програмування загального призначення, розроблена для задоволення потреб комп'ютерників, розробників програмного забезпечення та студентів коледжів, які цікавляться кодуванням. Python був створений на початку 1980-х Гвідо ван Россумом, коли він працював у IBM. Мова названа на честь її винахідника. Вона стала однією з найпопулярніших мов програмування в новітній історії. Python — це високорівнева, інтерпретована динамічна об'єктно-орієнтована мова, яку можна

використовувати для розробки програм або веб-сайтів. Основні переваги використання Python замість інших мов полягають у тому, що з ним дуже легко програмувати, а завдяки гнучкому синтаксису його можна використовувати для програмування майже будь-якого програмного забезпечення. Python — це мова з відкритим вихідним кодом, яка є безкоштовною для використання та має широкий спектр функцій, які спрощують її налаштування. Це також чудовий вибір для початківців, оскільки його використовувати. Окрім простоти використання, Python має низку інших функцій, які роблять його чудовим вибором для програмістів, які хочуть поринути у світ розробки програмного забезпечення.

Перш ніж заглибитися в основні функції Python, подивимося, що таке Python. Python був розроблений як наступник мови програмування ABC і вперше випущений у 1991 році як Python 0.9.0. Python є однією з найпопулярніших мов програмування сучасності. Python — це мова програмування високого рівня загального призначення, філософія розробки якої здебільшого наголошує на читабельності коду з використанням значного коду. Написання коду на Python як для малих, так і для великих проектів стає дуже простим навіть для сучасних програмістів-початківців, оскільки мовні конструкції та об’єктно-орієнтований підхід Python роблять код надзвичайно зрозумілим і логічним. Python підтримує різноманітні парадигми програмування, такі як об’єктно-орієнтоване програмування, структуроване програмування, функціональне програмування тощо, і має багатий набір бібліотек, таких як NumPy, Pandas тощо, що робить його чудовим вибором для різноманітних технічних галузей, таких як Data Science, машинне навчання тощо.

Python тепер доступний у версії 3.10. Він має відкритий код і його можна безкоштовно завантажити з офіційного веб-сайту [6].

Основні характеристики Python

1. Легка для вивчення та читання мова. Python надзвичайно простий у вивченні. Його синтаксис надзвичайно простий, а крива навчання Python дуже плавна. Його надзвичайно легко вивчати та кодувати на Python.

2. Інтерпретована мова. Python — це інтерпретована мова і IDLE упаковується разом із Python. Це не що інше, як інтерпретатор, який дотримується структури REPL (Read Evaluate Print Loop), як і в Node.js. IDLE виконує та відображає вивід одного рядка коду Python за раз. Таким чином, він відображає помилки, коли ми виконуємо рядок коду Python, і відображає всю трасування стека для помилки.

3. Динамічно типізована мова. Python є динамічно типізованою мовою. Іншими словами, у Python не потрібно оголошувати типи даних змінних, які ми визначаємо. Завданням інтерпретатора Python є визначення типів даних змінних під час виконання на основі типів частин виразу. Хоча це полегшує кодування для програмістів, ця властивість може спричинити помилки під час виконання. Точніше кажучи, Python слідує качиному введенню, це означає, що «якщо він виглядає як качка, плаває як качка і крякає як качка, це має бути качка».

4. Відкритий і безкоштовний. Python — це мова програмування з відкритим кодом, яку можна безкоштовно завантажити з офіційного веб-сайту Python. Спільнота користувачів Python постійно вносить свій внесок у код Python, щоб покращити його.

5. Велика стандартна бібліотека. Однією з дуже важливих особливостей, завдяки якій Python настільки відомий у наш час, є величезна стандартна бібліотека, яку він пропонує своїм користувачам. Стандартна бібліотека Python надзвичайно велика з різноманітним набором пакетів і модулів, таких як itertools, functools, operator та багато інших із загальними та важливими функціями в них. Якщо код певної функціональності вже присутній у цих модулях і пакетах, розробникам не потрібно переписувати їх з нуля, заощаджуючи час і зусилля з боку розробника.

6. Високорівнева мова. Високорівнева мова (high-level language (HLL)) — це мова програмування, яка дозволяє програмісту писати програми, більш-менш незалежні від певного типу комп’ютера. Ці мови вважаються високорівневими, оскільки вони дуже близькі до людських мов і далекі від машинних мов. На відміну від C, Python є мовою високого рівня. Можна легко зрозуміти Python, і

він ближчий до користувача, ніж мови середнього рівня, такі як C. У Python не потрібно пам’ятати архітектуру системи чи керувати пам’яттю.

7. Об'єктно-орієнтована мова програмування. Python підтримує різні парадигми програмування, однак найважливішим фактом є те, що об’єктно-орієнтований підхід Python дозволяє своїм користувачам реалізувати концепції інкапсуляції, наслідування, поліморфізму тощо, що надзвичайно важливо для кодування, яке виконується в більшості галузей програмного забезпечення.

8. Незалежність від платформи. Незалежність від платформи є ще однією дивовижною особливістю Python. Іншими словами, це означає, що якщо написати програму на Python, вона може працювати на різних платформах, наприклад, Windows, Mac, Linux тощо. Не потрібно писати окремий код Python для різних платформ.

9. Розширювані та вбудовані. Python — це мова, яку можна вбудовувати. Можна написати деякий код Python мовою C або C++, а також можна скомпілювати цей код мовою C/C++. Python також є розширюваним. Це означає, що можна розширити код Python різними мовами, такими як C++ тощо.

10. Підтримка графічного інтерфейсу користувача (GUI). Ще однією цікавою особливістю Python є те, що можна використовувати його для створення GUI (графічних інтерфейсів користувача). Можна використовувати Tkinter, PyQt, wxPython або Pyside, щоб зробити те саме. Python також має велику кількість графічних інтерфейсів, доступних для нього та різних інших кросплатформних рішень. Python прив’язується до платформо-специфічних технологій.

11. Спрощує розробку складного програмного забезпечення. Python можна використовувати для розробки як десктопних, так і веб-програм, а також складних наукових і числових програм. Функції аналізу даних Python допомагають створювати власні рішення для великих даних, не витрачаючи багато часу та зусиль. Також можна використовувати бібліотеки та API візуалізації даних Python, щоб представити дані більш привабливим способом.

Кілька передових розробників програмного забезпечення використовують Python для виконання висококласних завдань штучного інтелекту [6].

Різниця між асинхронним і синхронним програмуванням

Синхронний і асинхронний — це два типи моделей програмування. Розуміння того, чим ці дві моделі відрізняються, має вирішальне значення для побудови API, створення архітектур на основі подій і вирішення завдань, що виконуються довго. Вибираючи, який метод і коли використовувати, важливо знати кілька ключових речей про синхронне та асинхронне програмування [7].

Асинхронне програмування

Асинхронне програмування є багатопоточною моделлю, яка найбільше підходить для мереж і комунікацій. Асинхронна — це неблокуюча архітектура, що означає, що вона не блокує подальше виконання, поки виконуються одна або кілька операцій.

За допомогою асинхронного програмування кілька пов’язаних операцій можуть виконуватися одночасно, не чекаючи виконання інших завдань. Під час асинхронного зв’язку сторони отримують і обробляють повідомлення, коли це зручно чи можливо, а не відповідають одразу після отримання.

Текстові повідомлення – це асинхронний спосіб спілкування. Одна особа може надіслати текстове повідомлення, а одержувач може відповісти на вільний час. Тим часом відправник може виконувати інші дії, очікуючи на відповідь.

Синхронне програмування

Синхронна архітектура відома як блокуюча архітектура і ідеально підходить для програмування реактивних систем. Як однопотокова модель, вона дотримується суворого набору послідовностей, що означає, що операції виконуються по черзі в ідеальному порядку. Під час виконання однієї операції вказівки інших операцій блокуються. Виконання першого завдання запускає наступне і так далі.

Щоб проілюструвати, як працює синхронне програмування, можна згадати про телефон. Під час телефонної розмови одна людина говорить, інша слухає. Коли перший закінчує, другий прагне відповісти негайно.

Випадки використання

Програмування змушує цифровий світ працювати, але без правильного поєднання програм і операцій виникне хаос і поганий досвід користувачів. Якщо операції будуть неналежним чином покладатися на асинхронне програмування, цифровий світ може перетворитися на божевільне, гіперактивне божевілля. І якщо операції будуть неналежним чином покладатися на синхронне програмування, цифровий світ може повністю зупинитися. Важливо розуміти, коли використовувати кожен тип програмування.

Коли використовувати асинхронний тип

Асинхронне програмування слід використовувати лише для програмування незалежних завдань, де воно відіграє вирішальну роль. Наприклад, асинхронні програми ідеально підходять для проектів розробки з великою кількістю ітерацій. Оскільки кроки не обов'язково слідувати фіксованій послідовності, асинхронне програмування дозволяє розвиватися вперед.

Чуйний інтерфейс користувача є чудовим варіантом використання для асинхронного планування. Візьмемо, наприклад, додаток для покупок. Коли користувач відкриває своє замовлення, розмір шрифту має збільшуватися. Замість того, щоб спочатку чекати завантаження історії та оновлення розміру шрифту, асинхронне програмування може змусити обидві дії виконуватися одночасно.

Коли використовувати синхронний тип

Асинхронне програмування відносно складне. Це може надто ускладнити речі та ускладнити читання коду. Синхронне програмування, з іншого боку, досить просте; його код легше писати і не вимагає відстеження та вимірювання потоків процесів (як це робить асинхронний). Оскільки завдання залежать одне від одного, необхідно знати, чи можуть вони виконуватися незалежно, не заважаючи одне одному.

Синхронне програмування може підійти, наприклад, для програми для покупок. Під час оформлення замовлення в Інтернеті користувач бажає купити всі товари разом, а не окремо. Замість виконання замовлення кожного разу, коли

користувач додає щось у свій кошик, синхронне програмування гарантує, що спосіб оплати та пункт призначення для всіх товарів вибираються одночасно.

Як вибрати між асинхронним і синхронним програмуванням

Вирішуючи, який підхід застосувати, може бути корисним вважати асинхронне програмування адаптивним, а синхронне програмування суворим. Асинхронне програмування — це багатозадачність, яка переходить від одного завдання до іншого та сповіщає систему про виконання кожного з них. Синхронне програмування функціонує як єдиний розум, перевіряючи одне завдання за раз у жорсткій послідовності.

Асинхронне програмування дозволяє одночасно виконувати більше завдань і зазвичай використовується для покращення взаємодії з користувачем, забезпечуючи легкий і швидкий потік завантаження.

Різниця між синхронним і асинхронним

Зрештою, вибір зводиться до операційних залежностей. Потрібно, щоб початок операції залежав від завершення іншої операції, чи потрібно, щоб вона запускалася незалежно?

Асинхронна архітектура не блокує, тому виконання одного завдання не залежить від іншого. Завдання можуть виконуватися одночасно. Синхронна — це блокуюча архітектура, тому виконання кожної операції залежить від завершення попередньої. Кожне завдання вимагає відповіді перед переходом до наступної ітерації [7].

Відмінності між асинхронним і синхронним включають:

- Асинхронний є багатопоточним, що означає, що операції або програми можуть виконуватися паралельно. Синхронний є однопотоковим, тому одночасно виконуватиметься лише одна операція чи програма.
- Асинхронний не блокує, що означає, що він надсилатиме кілька запитів на сервер. Синхронізація блокується — вона надсилатиме серверу лише один запит за раз і чекатиме, доки сервер відповість на цей запит.

- Асинхронний збільшує пропускну здатність, оскільки кілька операцій можуть виконуватися одночасно. Синхронний повільніший та більш методичний.

Крім відмінностей, асинхронний і синхронний методи мають переваги, але для різних зацікавлених сторін: асинхронний для користувачів, синхронний для розробників.

Асинхронні та синхронні методи пропонують переваги для різних зацікавлених сторін; асинхронний для користувачів, синхронний для розробників.

Асинхронне програмування покращує роботу користувача, зменшуючи час затримки між викликом функції та поверненням значення цієї функції. У реальному світі це означає швидший і безперебійний потік. Наприклад, користувачі хочуть, щоб їхні програми працювали швидко, але потрібен час, щоб отримати дані з API. У цих випадках асинхронне програмування допомагає швидше завантажувати екрани програми, покращуючи взаємодію з користувачем.

З іншого боку, синхронне програмування є вигідним для розробників. Простіше кажучи, синхронне програмування набагато легше кодувати. Він добре підтримується серед усіх мов програмування, і як метод програмування за замовчуванням розробникам не потрібно витрачати час на вивчення чогось нового.

1.2.2 Python як інструмент створення API

Спеціалістам або інженерам з обробки даних часто доводиться ділитися своєю роботою, щоб будь-хто інший міг використовувати створені нами процеси чи моделі. Зрозуміло, що ділитися сценарієм не можна, оскільки всі повинні мати ті самі програми. Саме тоді в гру вступають API [8].

Основи API

API дозволяє двом комп’ютерним системам взаємодіяти одна з одною. Наприклад, якщо створюється автоматизація, яка генерує звіт і надсилає його

електронною поштою, надсилання цього електронного листа не виконується вручну, це зробить сам сценарій. Для цього Python (або мова, яку використовують), має попросити Gmail надіслати цей електронний лист із прикріпленим звітом для певних людей. Це можна зробити за допомогою API, у даному випадку API Gmail.

Основні частини API [8]:

1. Протокол передачі HTTP: це основний спосіб передачі інформації в Інтернеті. Існують різні методи, кожен з яких використовується для різних проблем:
   - GET: цей метод дозволяє отримати інформацію з бази даних або з процесу.
   - POST: дозволяє надсилати інформацію, наприклад, щоб додати інформацію до бази даних або передати вхідні дані моделі машинного навчання.
   - PUT: оновити інформацію. Зазвичай він використовується для керування інформацією в базі даних.
   - DELETE: цей метод використовується для видалення інформації з бази даних.
2. Url: це адреса, де ми можемо знайти наш API. В основному ця URL-адреса складатиметься з трьох частин:
   - Протокол: як і будь-яка адреса, це може бути http:// або https://.
   - Домен: хост, на якому він розміщений, який йде від протоколу до кінця .com або будь-якого закінчення URL-адреси.
   - Кінцева точка: як веб-сайт має кілька сторінок (/ blog), (/ legal), той самий API може включати кілька точок, і кожна з них виконує різні дії. Створюючи API на Python, вказуватимуться кінцеві точки, тому потрібно переконатися, що кожна точка відповідає тому, що робить API, що стоїть за нею.

Створення API запитів у Python

Щоб працювати з API в Python, нам потрібні інструменти, які будуть робити ці запити. У Python найпоширенішою бібліотекою для створення запитів і роботи з API є бібліотека requests. Бібліотека requests не є частиною стандартної бібліотеки Python, тому її потрібно встановити, щоб почати роботу і імпортувати.

Існує багато різних типів запитів. Найбільш часто використовуваний запит GET використовується для отримання даних. Коли робиться запит, відповідь від API надходить із кодом відповіді, який повідомляє, чи був запит успішним. Коди відповіді важливі, оскільки вони негайно повідомляють, якщо щось пішло не так.

Щоб зробити запит GET, використовується функція requests.get(), для якої потрібен один аргумент — URL-адреса, на яку потрібно зробити запит.

Веб-фреймворк Django

Django — це високорівневий веб-фреймворк Python, який забезпечує швидку розробку безпечних веб-сайтів, які зручно підтримувати. Створений досвідченими розробниками, Django бере на себе більшу частину клопоту веб-розробки. Він безкоштовний із відкритим кодом, має процвітаючу та активну спільноту, чудову документацію та багато варіантів безкоштовної та платної підтримки [9].

Django допоможе вам написати програмне забезпечення, яке:

1. Повний. Django надає майже все, що розробники можуть захотіти зробити. Оскільки все, що потрібно, є частиною одного «продукту», все це бездоганно працює разом, дотримується послідовних принципів проектування та має розширену та актуальну документацію.
2. Універсальний. Django можна використовувати для створення майже будь-якого типу веб-сайту — від систем керування контентом і вікі до соціальних мереж і сайтів новин. Він може працювати з будь-яким клієнтським фреймворком і може надавати вміст майже в будь-якому форматі (включаючи HTML, RSS-канали, JSON і XML).
3. Безпечний. Django допомагає розробникам уникати багатьох поширених помилок безпеки, надаючи структуру, розроблену так, щоб

«робити правильні речі» для автоматичного захисту веб-сайту. Наприклад, Django забезпечує безпечний спосіб керування обліковими записами та паролями користувачів, уникаючи поширених помилок, таких як розміщення інформації про сеанс у файлах cookie, де вони вразливі або пряме збереження паролів, а не хеш пароля.

4. Масштабований. Django використовує компонентну архітектуру "спільного використання нічого" (кожна частина архітектури не залежить від інших і, отже, може бути замінена). Наявність чіткого розподілу між різними частинами означає, що його можна масштабувати для збільшення трафіку шляхом додавання обладнання на будь-якому рівні: сервери кешування, сервери баз даних або сервери додатків. Деякі з найбільш завантажених сайтів успішно масштабували Django відповідно до своїх потреб.

5. Портативний. Django написано мовою Python, яка працює на багатьох платформах. Це означає, що немає прив’язки до жодної конкретної серверної платформи та можна запускати свої програми на багатьох версіях Linux, Windows і macOS. Крім того, Django добре підтримується багатьма провайдерами веб-хостингу, які часто надають спеціальну інфраструктуру та документацію для розміщення сайтів Django.

6. Досяжний. Код Django написаний з використанням принципів дизайну та шаблонів, які заохочують створення коду, який можна підтримувати та використовувати повторно. Зокрема, він використовує принцип «Don't Repeat Yourself» (DRY), тому немає непотрібного дублювання. Django також сприяє групуванню пов’язаних функціональних можливостей у багаторазово використовувані «додатки» та, на нижчому рівні, групує пов’язаний код у модулі (за зразком Model View Controller (MVC)).

MVC (Model-View-Controller) — це патерн у розробці програмного забезпечення, який зазвичай використовується для реалізації інтерфейсів

користувача, даних і логіки керування. Це підкреслює розмежування між бізнес-логікою програмного забезпечення та дисплеєм. Такий «розподіл завдань» забезпечує кращий розподіл праці та поліпшення технічного обслуговування. Деякі інші шаблони проектування засновані на MVC, наприклад MVVM (модель-вид-вид-модель), MVP (модель-вид-представник) і MVW (модель-вид-що завгодно) [9].

Три частини шаблону проектування програмного забезпечення MVC можна описати наступним чином:

- Модель: керує даними та бізнес-логікою.
- Перегляд: керує макетом і відображенням.
- Контролер: направляє команди до частин моделі та виду.

Django REST framework (DRF) — це потужний і гнучкий інструментарій для створення веб-API. Його головна перевага полягає в тому, що він значно полегшує серіалізацію.

Фреймворк Django REST базується на представленнях Django на основі класів, тому це чудовий варіант, якщо ви знайомі з Django. Він використовує такі реалізації, як перегляди на основі класів, форми, валідатор моделі, QuerySet тощо.

Веб-фреймворк FastAPI

Створення API або інтерфейсів програмування додатків є важливою частиною того, щоб зробити програмне забезпечення доступним для широкого кола користувачів [10].

FastAPI — це відносно новий асинхронний веб-фреймворк для Python, по суті, це гібрид Starlett (асинхронний веб-фреймворк) та Pydantic (бібліотека для валідації даних, серіалізації тощо); це сучасна, високопродуктивна веб-платформа для створення API за допомогою Python на основі стандартних підказок типу Python. Має такі ключові особливості [11]:

- Швидкий запуск: пропонує дуже високу продуктивність, нарівні з NodeJS і Go, завдяки Starlette і Pydantic. Один із найшвидших фреймворків Python.
- Швидке кодування: дозволяє значно збільшити швидкість розробки.

- Менше помилок: зменшує ймовірність помилок, спричинених людиною (розробником).
- Інтуїтивно зрозумілий. Пропонує чудову підтримку редактора Менше часу на налагодження.
- Простий: розроблений, щоб бути простим у використанні та навчанні. Менше часу на читання документів.
- Короткий: зводить до мінімуму дублювання коду.
- Надійний: забезпечує готовий до виробництва код із автоматичною інтерактивною документацією.
- На основі стандартів: базується (і повністю сумісний) з відкритими стандартами для API: OpenAPI і JSON Schema.

Аргументація вибору інструментарію розробки

Для реалізації було обрано веб-фреймворк FastAPI з декількох причин:

1. JSON – максимально уніфікований формат даних зв`язку фронтенду та бекенду. FastAPI реалізовує RESTful протокол, тому має в основі обмін саме json даними між ендпоїнтами.
2. Можливість реалізації асинхроних функцій-обробників, що дозволятиме паралельно виконувати декілька запитів на різні ендоїнти, не очікуючи закінчення процесу обробки іншими ендпоїнтами.
3. Швидкість розробки, оскільки для обробників не є необхідним створення додаткових класів (як потрібно в Django).
4. Бібліотека Pydantic в основі валідації, що дозволяє створювати уніфіковані моделі даних, які в подальшому можна буде також використати в ORM-режимі.

1.2.3 Python як мова реалізації алгоритмів машинного навчання

Машинне навчання – це метод аналізу даних, який автоматизує створення аналітичної моделі. Це галузь штучного інтелекту, заснована на ідеї, що системи можуть навчатися на даних, ідентифікувати закономірності та приймати рішення з мінімальним втручанням людини.

Python є найпопулярнішою платформою, яка використовується для дослідження та розробки виробничих систем. У ньому є кілька модулів, пакетів і бібліотек, які надають різні способи виконання завдань в машинному навчанні. Різні бібліотеки в Python широко використовуються для створення масштабованих алгоритмів машинного навчання. Python пропонує готову структуру для ефективного виконання завдань інтелектуального аналізу даних на великих обсягах даних за менший час [12].

Python має чудову бібліотечну екосистему, яка пропонує різноманітні інструменти. Основні з них, які використовуються в машинне навчання:

- SciKit-learn - для роботи з класичними алгоритмами машинного навчання;
- Pandas - використовується для високорівневих структур даних і аналізу;
- Keras і TensorFlow - для аналізу глибинного навчання;
- Matplotlib — бібліотека двовимірних графіків для створення графіків і графіків для візуалізації;
- NumPy – використовується для об’єктів N-вимірного масиву

Scikit-learn (Sklearn) — це найкорисніша та надійна бібліотека для машинного навчання на Python. Він надає вибір ефективних інструментів для машинного навчання та статистичного моделювання, включаючи класифікацію, регресію, кластеризацію та зменшення розмірності через узгоджений інтерфейс у Python. Ця бібліотека, яка в основному написана на Python, побудована на NumPy, SciPy і Matplotlib.

Особливості бібліотеки Scikit-learn

Замість того, щоб зосереджуватися на завантаженні, маніпулюванні та узагальненні даних, бібліотека Scikit-learn зосереджена на моделюванні даних. Деякі з найпопулярніших груп моделей, наданих Sklearn, такі [13]:

1. Алгоритми керованого навчання. Майже всі популярні алгоритми керованого навчання, як-от лінійна регресія, дерево рішень тощо, є частиною scikit-learn.
2. Кластеризація. Ця модель використовується для групування немаркованих даних.

3. Перехресна перевірка. Використовується для перевірки точності контрольованих моделей на невидимих даних.

4. Зменшення розмірності – використовується для зменшення кількості атрибутів у даних, які в подальшому можна використовувати для підсумовування, візуалізації та вибору ознак.

5. Методи ансамблю. Він використовується для об’єднання прогнозів кількох контрольованих моделей.

6. Вилучення функцій. Використовується для вилучення ознак із даних для визначення атрибутів у даних зображення та тексту.

7. Вибір функцій. Використовується для визначення корисних атрибутів для створення контрольованих моделей.

8. Open Source. бібліотека з відкритим вихідним кодом, яка також може використовуватися в комерційних цілях.

1.2.4 Робота з зовнішніми сервісами

SQLAlchemy та Alembic для роботи з БД

SQLAlchemy — це набір інструментів Python SQL і Object Relational Mapper, який надає розробникам додатків повну потужність і гнучкість SQL.

Основні функції SQLAlchemy включають [14]:

1. Промислова потужність ORM, створена з ядра на карті ідентичності, одиниці роботи та шаблонах відображення даних. Ці шаблони дозволяють прозоро зберігати об’єкти за допомогою декларативної системи конфігурації. Доменні моделі можна створювати та керувати ними природним шляхом, а зміни синхронізуються з поточною транзакцією автоматично.

2. Реляційно-орієнтована система запитів, яка явно розкриває повний спектр можливостей SQL, включаючи об’єднання, підзапити, кореляцію та багато іншого, з точки зору об’єктної моделі. Написання запитів за допомогою ORM використовує ті самі методи реляційної композиції, які

ви використовуєте під час написання SQL. Хоча ви можете будь-коли перейти до буквального SQL, він практично ніколи не потрібен.

3. Комплексна та гнучка система швидкого завантаження пов'язаних колекцій та об'єктів. Збірки зберігаються в кеш-пам'яті протягом сеансу та можуть бути завантажені з індивідуальним доступом, усі одночасно за допомогою об'єднань або за запитом на колекцію в повному наборі результатів.

4. Конструкційна система Core SQL і рівень взаємодії DBAPI. Ядро SQLAlchemy є відокремленим від ORM і є власним рівнем повної абстракції бази даних і включає розширювану мову виразів SQL на основі Python, метадані схеми, пул з'єднань, приведення типів і спеціальні типи.

5. Усі обмеження первинного та зовнішнього ключів вважаються складеними та природними. Сурогатні цілі первинні ключі, звичайно, все ще є нормою, але SQLAlchemy ніколи не припускає і не жорстко кодує цю модель.

6. Інтроспекція та генерація бази даних. Схеми баз даних можуть бути «відображені» за один крок у структурах Python, що представляють метадані бази даних; ті самі структури можуть потім генерувати оператори CREATE відразу ж назад - все в межах ядра, незалежно від ORM.

Alembic — це інструмент для міграції бази даних, написаний автором SQLAlchemy. Інструмент міграції пропонує такі функції [15]:

- Може видавати оператори ALTER (використовується для додавання, видалення або зміни стовпців у існуючій таблиці.) до бази даних, щоб змінити структуру таблиць та інших конструкцій
- Надає систему, за допомогою якої можна створювати «сценарії міграції»; кожен сценарій вказує на певну серію кроків, які можуть «оновити» цільову базу даних до нової версії, і, за бажанням, серію кроків, які можуть «понизити» подібним чином, виконуючи ті самі кроки у зворотному порядку.
- Дозволяє сценаріям виконуватися певним чином.

Пакети з класами-конекторами для БД Redis

Redis — це мережеве сховище даних із відкритим вихідним кодом, що знаходиться в пам’яті та має додаткову довговічність.

Початок роботи з Redis

redis-py — клієнт Python, який дозволяє спілкуватися з сервером Redis через зручний Python виклик:

```
$ python -m pip install
```

Далі потрібно переконатися, що сервер Redis все ще запущений у фоновому режимі. Це можна зробити за допомогою *pgrep redis-server*, і, якщо нічого немає, то потрібно перезапустити локальний сервер за допомогою *redis-server /etc/redis/6379.conf*.

Тепер частина орієнтована на Python:

```
>>> import redis
>>> r = redis.Redis()
>>> r.mset({"1": "2", "3": "4"})
True
>>> r.get("3")
b'4'
```

Redis, який використовується в рядку 2, є центральним класом пакету, за допомогою якого виконується (майже) будь-яка команду Redis. З’єднання та повторне використання TCP-сокетів виконується за лаштунками, і виклик команди Redis відбувається за допомогою методів екземпляра класу r.

Потрібно також зауважити, що типом повернутого об’єкту буде *bytes*.

Методи майже у всіх випадках збігаються з назвою команди Redis, яка виконує те саме. У прикладі вище було викликано r.mset() і r.get(), які відповідають MSET і GET у Redis API. Це також означає, що HGETALL стає r.hgetall(), PING стає r.ping() і так далі. Є кілька винятків, але правило діє для переважної більшості команд.

Хоча аргументи команди Redis зазвичай перетворюються на схожу на вигляд підпис методу, вони беруть об’єкти Python. Наприклад, виклик r.mset() у прикладі вище використовує dict Python як перший аргумент, а не послідовність байтових рядків.

У прикладі вище було створено екземпляр Redis r без аргументів, але він поставляється в комплекті з низкою параметрів, якщо вони потрібні:

```
# From redis/client.py
class Redis(object):
    def __init__(self, host='localhost', port=6379,
                 db=0, password=None, socket_timeout=None,
                 # ...
```

Можна побачити, що пара ім’я хоста:порт за замовчуванням localhost:6379, і це саме те, що потрібно у випадку локально збереженого екземпляра redis-server.

Параметр db — це номер бази даних. Можна керувати декількома базами даних у Redis одночасно, і кожна ідентифікується цілим числом. За замовчуванням максимальна кількість баз даних становить 16 [16].

Інструменти деплойемнту та разгортання сервісу

Uvicorn — це реалізація веб-сервера ASGI (Asynchronous Server Gateway Interface) для Python. До недавнього часу в Python був відсутній мінімальний низькорівневий інтерфейс сервера/програми для асинхронних фреймворків. Специфікація ASGI заповнює цю прогалину і означає, що тепер ми можемо почати створювати загальний набір інструментів, які можна використовувати в усіх асинхронних фреймворках [17].

HTTP — протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від HyperText Transfer Protocol [18].

HTTPS — схема URI, що синтаксично ідентична http: схемі, яка зазвичай використовується для доступу до ресурсів Інтернет. Використання https: URL вказує, що протокол HTTP має використовуватися, але з іншим портом за замовчуванням і додатковим шаром шифрування/автентифікації між HTTP і TCP [19].

Проксі-сервер — це проміжний сервер, який отримує дані з Інтернет-джерела, наприклад веб-сторінки, від імені користувача. Вони діють як додаткові межі безпеки даних, захищаючи користувачів від зловмисної діяльності в Інтернеті [20].

Проксі-сервери мають багато різних застосувань, залежно від їх конфігурації та типу. Загальні способи використання включають полегшення анонімного перегляду Інтернету, обхід геоблокування та регулювання веб-запитів.

Як і будь-який пристрій, підключений через Інтернет, проксі пов'язані з ризиками кібербезпеки, які користувачі повинні враховувати перед використанням.

1.3 Лінійна регресія як інструмент для підбору відношень сутностей

Лінійний регресійний аналіз використовується для прогнозування значення змінної на основі значення іншої змінної. Змінна, яку потрібно передбачити, називається залежною змінною. Змінна, яка використовується для прогнозування значення іншої змінної, називається незалежною змінною [21].

Ця форма аналізу оцінює коефіцієнти лінійного рівняння, що включає одну або кілька незалежних змінних, які найкраще передбачають значення залежної змінної. Лінійна регресія відповідає прямій лінії або поверхні, що мінімізує розбіжності між прогнозованими та фактичними вихідними значеннями. Існують прості калькулятори лінійної регресії, які використовують метод найменших квадратів для виявлення найкращої лінії для набору парних даних. Потім оцінюється значення X (залежна змінна) з Y (незалежна змінна).

Можна виконувати лінійну регресію в Microsoft Excel або використовувати пакети статистичних програм, такі як IBM SPSS® Statistics, які значно спрощують процес використання рівнянь лінійної регресії, моделей лінійної регресії та формули лінійної регресії. SPSS Statistics можна використовувати в таких методах, як проста лінійна регресія та множинна лінійна регресія.

Можна виконувати метод лінійної регресії в різних програмах і середовищах, зокрема:

- R лінійна регресія;
- лінійна регресія MATLAB;
- лінійна регресія Sklearn;
- лінійна регресія Python;
- лінійна регресія Excel;

Лінійна регресія в Python. Для цього застосовуються відповідні бібліотеки та їхні функції і класи [22].

NumPy — це фундаментальний науковий пакет Python, який дозволяє багато високопродуктивних операцій над одновимірними та багатовимірними масивами. Він також пропонує багато математичних процедур. Звичайно, це з відкритим кодом.

Пакет scikit-learn — це широко використовувана бібліотека Python для машинного навчання, побудована на основі NumPy та деяких інших пакетів. Він надає засоби для попередньої обробки даних, зменшення розмірності, впровадження регресії, класифікації, кластеризації тощо. Як і NumPy, scikit-learn також має відкритий код.

Якщо потрібно реалізувати лінійну регресію та є потреба в функціональності, що виходить за рамки scikit-learn, слід розглянути statsmodels. Це потужний пакет Python для оцінки статистичних моделей, виконання тестів тощо.

Висновки до розділу

Автором було вивчено теоретичну базу наукового дослідження, проведено порівняльний аналіз можливих засобів реалізації поставлених задач та на основі отриманих даних інструментами реалізації було обрано:

- мова програмування Python 3.10;
- синхронний веб-фреймворк FAST API, який реалізує RESTful протоколи;

- для реалізації алгоритмів машинного навчання було обрано модель лінійної регресії, яку в подальшому буде імплементовано за допомогою бібліотеки ScikitLearn.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ДОДАТКУ «WORK SEARCH API»

2.1 Концептуальна модель

«Work search API» - це веб-додаток, який реалізує концепцію REST API, що представляє собою набір ендопоїнтів для маніпулювання даними. API має реалізовувати можливості використання основних CRUD (Create-Read-Update-Delete) операцій для сутностей та надавати функціонал для створення відношення сутностей Резюме та Вакансія один до одного з метою підбору найбільш вдалих співпадінь критеріїв відношення.

Фактично додаток буде представляти собою набір функцій-обробників HTTP запитів типів GET, POST, PATCH та DELETE та додаткові функції для реалізації алгоритмів машинного навчання, інтегровані в основну систему.

2.1.1 REST API як бекенд веб-додатку

Основою веб-серверу обробки запитів буде RESTful API, який маніпулює даними формату JSON (Javascript Object Notation). API має реалізовувати основні принципи REST архітектури:

1. складатися з клієнтів, серверів і ресурсів із запитами, керованими через HTTP;
2. інформація про клієнта має не зберігається між запитами на отримання, і кожен запит бути окремим і не пов’язаним;
3. реалізовувати кешування даних;
4. форма передачі даних має бути уніфікованою, в даному випадку JSON об’єктами
5. обробка запитів має проходити невидимо для клієнту і проходити всі рівні серверів (відповідальних за безпеку, балансування навантаження тощо).

2.1.2 Алгоритм машинного навчання для підбору резюме під вакансію

Алгоритм має складатися з чотирьох основних кроків:

1. створення моделі лінійної регресії для кожної нової вакансії після її внесення до бази даних (підготовчий крок, який має виконуватися автоматично при створенні вакансії);
2. тренування моделі на основі розставлених пріоритетів навичок для визначення коефіцієнту сумісності (підготовчий крок, який має виконуватися автоматично при створенні вакансії);
3. обрання сумісних вакансій, у яких наявно мінімум три обов'язкових навички;
4. передбачення сумісності резюме до заданої вакансії на основі натренованої моделі.

Для передбачення буде використана лінійна регресія, для імплементації (впровадження) якої буде створено алгоритм приведення інформації про навички до єдиного числа, яке позначимо як «коефіцієнт навичок». Після чого, значення цього коефіцієнту буде передано у вже натреновану модель. У результаті отримаємо процентну сумісність резюме з вакансією.

2.1.3 Алгоритм машинного навчання для підбору вакансій під резюме

Алгоритм підбору вакансій до резюме працюватиме по аналогії з вищеописаним. Основною відмінністю буде модель даних, якою буде оперувати лінійна регресія.

Виходячи з цього, для попередження дуплікації коду, функцію-реалізацію алгоритму потрібно імплементувати (впровадити) таким чином, щоб на вхід, як цільові, можна було подавати обидві моделі – Vacancy та Resume.

2.1.4 Вимоги до бази даних

Базу даних необхідно створити з огляду на збереження та обробку інформації про вакансію, резюме, компанію та користувача-шукача роботи, також потрібно, враховуючи різні рівні доступу, забезпечити збереження ролей

кожного з користувачів. Надати доступи на редагування та видалення вакансій тільки компаніям, які їх створили. Аналогічно і з даними резюме.

Окремо необхідно створити допоміжні таблиці для навичок, мов програмування, які будуть використані в створені вакансій та резюме в подальшому.

2.1.5 Опис вхідних даних

В залежності від того, хто використовує дану програму (користувач чи компанія) можливо два потоки вхідних даних.

У випадку, якщо доступ отримав користувач, він матиме змогу створити свій профіль та своє резюме. В такому випадку, йому потрібно забезпечити API даними, відображеними на рис.2.1.

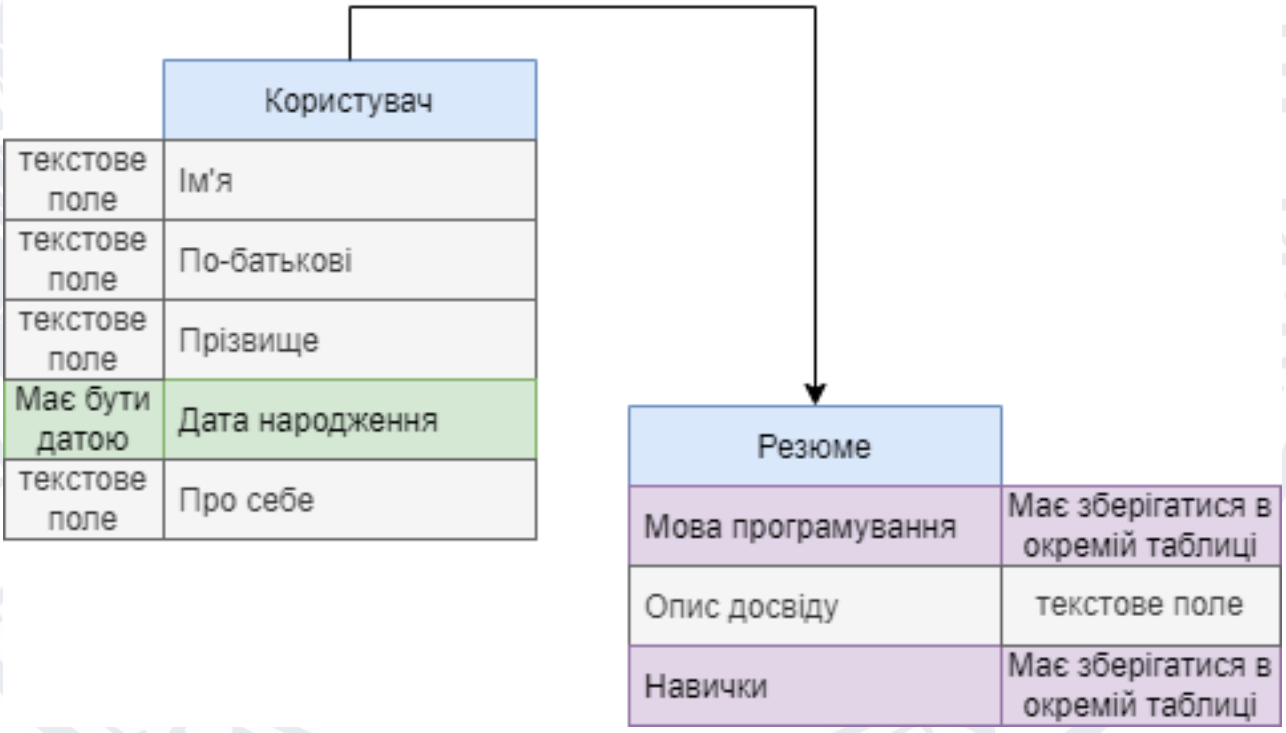

Рисунок 2.1 — дані для API з боку користувача

Компанія може створити дві сутності – профіль компанії та вакансію. Для цього потрібно буде надати дані, відображенні на рис. 2.2.

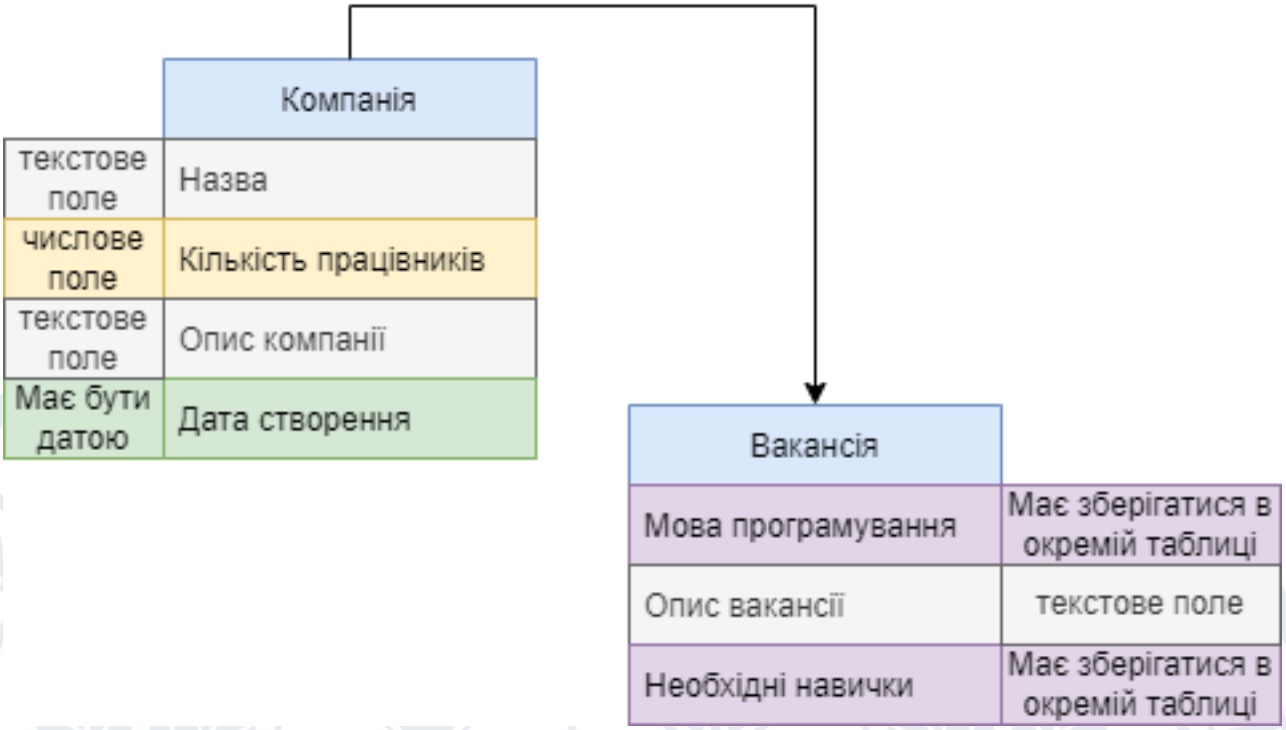

Рисунок 2.2 — дані для API з боку компанії

Також кожна з представлених сутностей має мати дані про дату створення та дату внесення останніх модифікацій. Також має бути передбачене поле для встановлення дати видалення сутності, якщо буде обрано “soft delete” метод видалення (тобто додання часової мітки про видалення, без самого стирання даних з бази).

Окремо, адміністраторами API мають бути створені сутності «Мова програмування» та «Навичка», в яку додані основні дані. Користувачам та компаніям має бути надана можливість внесення нових даних, однак редагування та видалення мають бути дозволені лише адміністраторам.

2.1.6 Вимоги до функціоналу

Розроблюване API має надавати наступний функціонал:

1. Робота з даними компаній (створення, редагування, читання та видалення)
2. Робота з даними вакансій (створення, редагування, читання та видалення)

3. Робота з даними користувачів (створення, редагування, читання та видалення)
4. Робота з даними резюме (створення, редагування, читання та видалення)
5. Аутентифікація HTTP запитів за допомогою унікальних ключів
6. Використання алгоритму підбору резюме до вакансії
7. Використання алгоритму підбору вакансій до резюме

2.2 Логічна модель

Програмний продукт має містити чотири основні сутності: компанія, вакансія, користувач і резюме.

Сутність «Компанія» відповідає за маніпуляції даними конкретної компанії, яка бажає створити вакансію на основі API. Дані мають бути надані представниками компанії.

Сутність «Вакансія» має бути створена лише під обліковим записом компанії, щоб мати коректне відношення між сутностями. Дані заповнюються представниками компанії.

Аналогічно мають працювати сутності «Користувач» та «Резюме», однак доступ до створення нового користувача має бути вільним (здійснюватися з вказанням API-key у заголовках HTTP-запиту однак без особливих ролей)

Дві сутності – «Вакансія» та «Резюме» мають також бути сумісними для використання у алгоритмах машинного навчання для підбору відношень між ними. Доступ до використання функцій підбору мають мати тільки користувачі, які створювали резюме/вакансію.

Програма має бути розділена на основні модулі, які будуть відповідати за опис та імплементацію моделей даних та алгоритмів. ORM-класи, які описують сутності баз даних мають зберігатися в модулі “models”. Класи, які інкапсулюють в собі основні операції з сутностями бази даних (CRUD-класи) мають бути розподілені в окремих файлах, в залежності від сутності з якою працюють, та винесені в модуль “crud”. Функції-обробники HTTP запитів також

мають бути розділені за сутністю вхідних даних та згруповані у модулі “routers”. Також передбачаються модулі “core” (основні константи та функції, що використовуються у інших модулях), “db” (класи-конектори до баз даних, класи сесій), “utils” (утиліти для роботи роутерів, алгоритми машинного навчання).

Окремо в корені репозиторію мають бути створені теки “alembic_migrations” та “tests”, в яких будуть знаходитися файли міграцій БД та файли з юніт- і інтеграційними тестами модулів відповідно.

2.3 Фізична модель

Для створення серверної частини використовується FastAPI фреймворк мови програмування Python, модуль SQLAlchemy для об’єктно-реляційного відображення. Деплоймент та запуск серверу здійснюється за допомогою модулю uvicorn.

Першим кроком зроблена конфігурація проекту, а саме підключення бази даних PostgreSQL для зберігання сутностей, та база даних HSQLDB, яка міститься в пам’яті, для пришвидшення виконання налагодження проекту, модульного та інтеграційного тестування на етапі розробки. Проведено конфігурацію бази даних Redis для зберігання ключів доступу API та здійснення аутентифікації та авторизації.

У даному програмному забезпечені використовується мережева топологія – зірка. Переваги:

- при виході з ладу одного з кабелів, з'єднання обірветься тільки одному користувачеві;
- простий пошук несправностей і обривів в мережі;
- простота перепідключення комп'ютерів і підключення нових користувачів;
- висока продуктивність мережі і гнучкі можливості адміністрування.

Дана мережа була обрана виходячи з попередніх пунктів. У її створенні використовуються такі засоби:

- Маршрутизатор;

- Мережевий кабель Ethernet;
- Комутатор;
- Мережевий кабель SFP;
- Сервер контролю домену;
- ПК.

Висновки до розділу

Під час роботи над розділом було створено актуальну модель майбутнього бекенду веб-додатку, розроблено основні алгоритми машинного навчання, які будуть в подальшому імплементовані, прописано основні структури даних, які використовуватиме REST API, визначені та сформульовані вимоги до функціоналу Application Programming Interface. Після визначення основних концепцій інформації, якою буде маніпулювати додаток, було побудовано логічну модель розроблюваного застосунку та визначені основні елементи фізичної моделі.

РОЗДІЛ 3
ПРОЕКТУВАННЯ БАЗИ ДАНИХ

3.1 Опис концептуальної моделі

Проектування бази даних полягає в побудові комплексу взаємозв'язаних моделей даних. Концептуальне моделювання дозволяє врахувати логічне уявлення структури даних у базі даних. Правильно розроблена модель бази даних має підтримувати усі явлення користувачів. Концептуальне моделювання є основою подальшого проектування бази даних та додатку для її обробки.

3.1.1 Вимоги до БД з боку користувачів

- Можливість зберігання великого обсягу даних
- Можливість створення облікових записів користувачів та компаній
- Можливість додавання нового резюме/вакансії, їх редагування та видалення (як soft, так і hard delete).

3.1.2 Визначення сутностей предметної області та їх аналіз

База даних складається з 4 основних сутностей: користувач, резюме, компанія, вакансія; та двох допоміжних сутностей – мова програмування, навичка.

Користувач ідентифікується унікальним номером (id) та має заповнити усі основні дані про себе – ім’я, по-батькові, прізвище, дата народження та опис свого досвіду. Таблиця має передбачати збереження часових міток для створення, оновлення та видалення користувача. У випадку використання процедури soft delete, як видалені мають позначатися усі резюме користувача.

Резюме є однією із головних сутностей БД, яка має зв’язки з обома допоміжними таблицями та має бути пов’язана з сутністю користувач зв’язком «багато-до-одного». Має передбачати поля з ключами до сутностей «мова програмування» (зі зв’язком «багато-до-одного») та «навичка» (зв’язок «багато-

до-багатьох»). Обов’язковими є поля з часовими мітками часу створення, оновлення та видалення резюме.

По аналогії з сутностями «користувач» та «резюме» мають бути створені сутності «компанія» та «вакансія».

Мова програмування – допоміжна сутність, яка має зберігати в собі інформацію про мову програмування необхідну у вакансії/резюме та її «коефіцієнт навички» (за замовчуванням 1).

Навичка – допоміжна сутність із інформацією про назву навички, з можливим заповненням опису основних критеріїв оцінки наявності навички та її коефіцієнтом (виставляється в залежності від пріоритету, наданого у вакансії).

3.1.3 Опис моделі проектованої системи у вигляді діаграми потоків даних

Опис моделі проектованої системи у вигляді діаграми потоків без декомпозиції поданий на наступному рис.3.1.

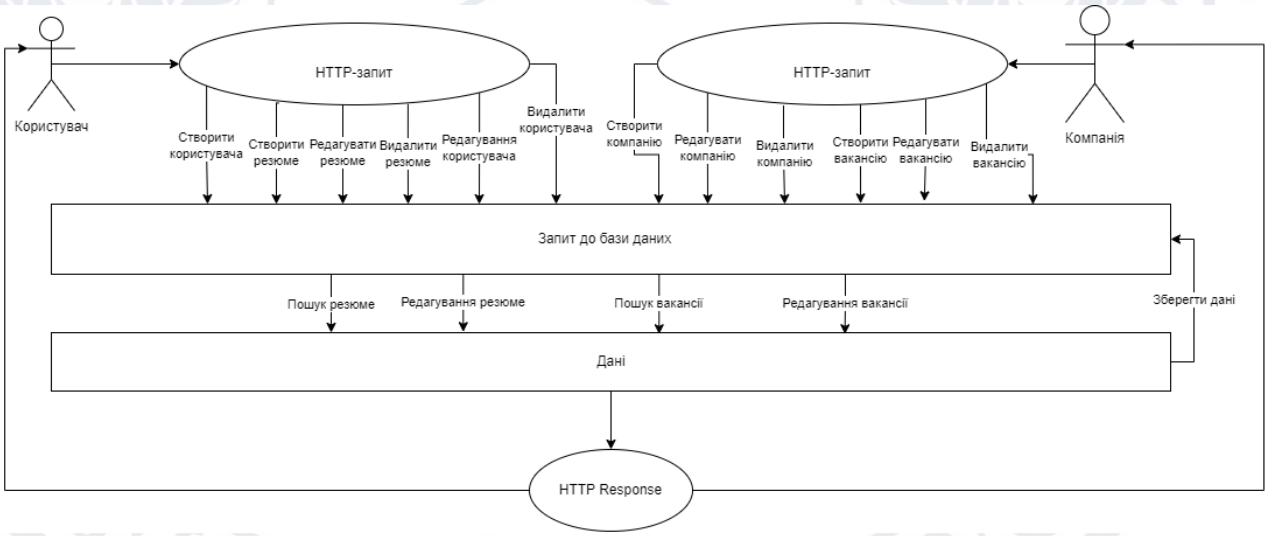

Рисунок 3.1 — діаграма потоків даних без декомпозиції

3.1.4 Опис моделі даних у вигляді ER-діаграми

На рис. 3.2 показана модель даних у вигляді ER-діаграми.

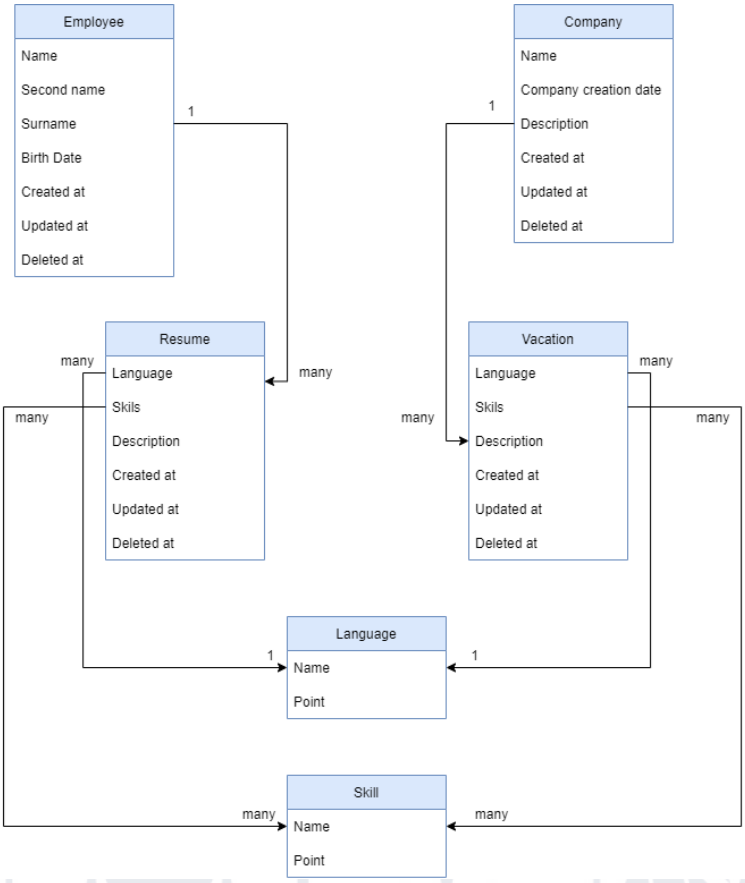

Рисунок 3.2 — модель даних у вигляді ER-діаграми

3.1.5 Опис обмежень

Основна база даних не має суттєвих обмежень, оскільки в основному використовується тип зв’язку між таблицями «1 до n».

Обмеження унікальності мають всі первинні ключі (що закладено в самому понятті «первинний ключ»). Усі первинні ключі мають тип BIGINT та обмеження по кількості символів – від 1 до 20. Інших обмежень унікальності немає.

3.2 Опис логічної моделі

Логічне проектування бази даних – це процес перетворення концептуальної моделі в логічну модель з урахуванням особливостей обраної СУБД. Основним завданням логічного проектування є розробка логічної схеми, орієнтованої на вибрану СУБД.

3.2.1 Опис набору відношень

Первинним ключем у всіх сутностях буде виступати їх номер – id.

Так як за вимогами реляційної моделі не можна використовувати на пряму зв’язки «багато до багатьох», то між такими таблицями потрібно створити, так звані, проміжні таблиці. Такі таблиці будуть мати у собі посилання на таблиці, між якими зв’язок «багато до багатьох» і таким чином зв’язок між проміжною таблицею та пов’язаною із нею буде «один до багатьох».

Основними сутностями програми є «Резюме» та «Вакансія». Основні набори зв’язків необхідно опрацьовувати відштовхуючись від них.

Резюме має зв'язок «n до 1» з сутністю «Користувач» - один користувач може створити багато резюме, однак кожне резюме має одного користувача; зв'язок «1 до n» з «Мовою програмування» та зв'язок «n до n» із сутністю навички. Остатній необхідно замінити на зв'язок «1 до n» до проміжної таблиці.

Аналогічні зв’язки має сутність «Вакансія».

3.2.2 Нормалізація до 3НФ

На рис. 3.3 показана БД, яка відповідає всім вимогам третьої нормальної форми баз даних.

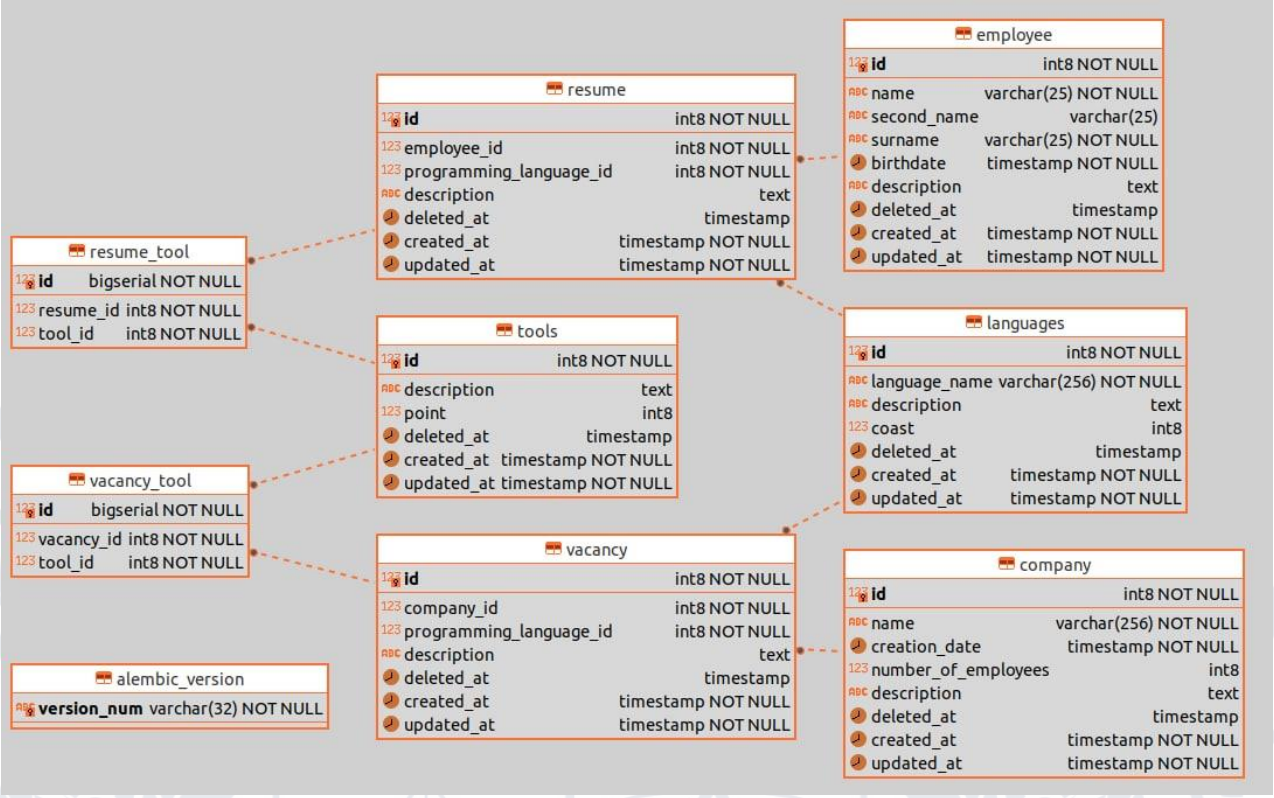

Рисунок 3.3 – схема бази даних

База даних знаходиться в 3НФ оскільки:

- Всі атрибути відношення є простими, всі використані домени містять тільки скалярні значення. Повторень рядків в таблиці немає.
- Відношення знаходиться в 1НФ і кожен не ключовий атрибут неприводимо залежить від Первинного Ключа (в складі потенційного ключа відсутня менша підмножина атрибутів, від якої можна також вивести дану функціональну залежність).
- Відношення знаходиться в 2НФ і кожен не ключовий атрибут нетранзитивно залежить від первинного ключа (все не ключові поля, вміст яких може стосуватися кількох записів таблиці, винесені в окремі таблиці).

3.3 Опис фізичної моделі

3.3.1 Вибір СУБД

Для обробки даних розробленої бази даних необхідно розробити додаток у MySQL.

Плюси:

- декларативність;
- простота коду запитів;
- наявність стандарту.

3.3.2 Опис таблиць

Умовно, таблиці, що складають всю БД, можна поділити на два типи: таблиці-словники, та таблиці з інформацією, що може часто змінюватися, або доповнюватись. Таблиці-словники найчастіше містять в собі статичну інформацію, яка не змінюється протягом довгого періоду часу. Найпростішим прикладом такої таблиці в даній базі даних будуть співвідношення «ключ – розшифрування».

Сутність Мова програмування

Короткий опис сутності. Сутність призначена для зберігання інформації про мову програмування.

Атрибути:

- id – ідентифікаційний код мови програмування
- language_name_varchar – назва мови
- description – опис мови
- coast – коефіцієнт
- deleted_at – часова мітка видалення мови
- creted_at – часова мітка створення мови
- updated_at – часова мітка оновлення мови

Зв’язки.

Одна чи більше мова програмування обов’язково міститься у вакансії.

Одна чи більше мова програмування обов’язково міститься у резюме.

Бізнес-правила. Атрибути language_name_varchar, creted_at та updated_at є обов’язковими.

Сутність Навичка

Короткий опис сутності. Сутність призначена для зберігання інформації про навички.

Атрибути:

- id – ідентифікаційний код навички
- description – опис основних критеріїв оцінки наявності навички
- coast – коефіцієнт навички
- deleted_at – часова мітка видалення навички
- creted_at – часова мітка створення навички
- updated_at – часова мітка оновлення навички

Зв'язки.

Одна чи більше навичка *обов'язково* міститься у резюме.

Одна чи більше навичка *обов'язково* міститься у вакансії.

Бізнес-правила. Атрибути creted_at та updated_at є обов'язковими.

Сутність Користувач

Короткий опис сутності. Сутність призначена для зберігання інформації про користувача.

Атрибути:

- id – ідентифікаційний код користувача
- name – ім'я користувача
- second name – по-батькові користувача
- surname – прізвище користувача
- birthdate – дата народження користувача
- description – опис власного досвіду користувача
- deleted_at – часова мітка видалення користувача
- creted_at – часова мітка створення користувача
- updated_at – часова мітка оновлення користувача

Зв'язки.

Користувач *може* мати *одне чи більше* резюме.

Бізнес-правила. Атрибути name, surname, birthdate, creted_at та updated_at є обов'язковими.

Сутність Резюме

Короткий опис сутності. Сутність призначена для зберігання інформації резюме.

Атрибути.

- id – ідентифікаційний код резюме
- emploee_id – ідентифікаційний код користувача
- programming_language_id – ідентифікаційний код мови програмування
- description – опис користувача
- deleted_at – часова мітка видалення резюме
- creted_at – часова мітка створення резюме
- updated_at – часова мітка оновлення резюме

Зв'язки.

Резюме *обов'язково* має мати *одного і тільки одного* користувача.

Резюме *обов'язково* має мати *одну і тільки одну* мову програмування.

Резюме *обов'язково* має мати *одну чи більше* навичку.

Бізнес-правила. Атрибути emploee_id, programming_language_id, creted_at та updated_at є обов'язковими.

Сутність Компанія

Короткий опис сутності. Сутність призначена для зберігання інформації про компанію.

Атрибути.

- id – ідентифікаційний код компанії
- name – назва компанії
- creation_date – дата створення компанії
- number_of_emploees – кількість працівників компанії
- description – опис компанії
- deleted_at – часова мітка видалення компанії

- creted_at – часова мітка створення компанії
- updated_at – часова мітка оновлення компанії

Зв'язки.

Компанія *обов'язково* має мати *одну чи більше* вакансію.

Бізнес-правила. Атрибути name, creation_date, creted_at та updated_at є обов'язковими.

Сутність Вакансія

Короткий опис сутності. Сутність призначена для зберігання інформації про вакансію.

Атрибути.

- id – ідентифікаційний код вакансії
- company_id – ідентифікаційний код компанії
- programming_language_id – ідентифікаційний код мови програмування
- description – опис вакансії
- deleted_at – часова мітка видалення вакансії
- creted_at – часова мітка створення вакансії
- updated_at – часова мітка оновлення вакансії

Зв'язки.

Вакансія *обов'язково* має мати *одну і тільки одну* компанію.

Вакансія *обов'язково* має містити *одну і тільки одну* мову програмування.

Бізнес-правила. Атрибути company_id, programming_language_id, creted_at та updated_at є обов'язковими.

Таблиця vacancy_tool

Короткий опис таблиці. Допоміжна таблиця для реалізації зв'зку «багато-до-багатьох» між сутностями Вакансія і Навичка.

Таблиця resume_tool

Короткий опис таблиці. Допоміжна таблиця для реалізації зв'зку «багато-до-багатьох» між сутностями Резюме і Навичка.

Таблиця alembic_version

Короткий опис таблиці. Таблиця містить в собі хеш-назву актуальної версії alembic-міграцій.

3.3.3 Опис типових SQL-запитів

- select - вибрати
- insert – ввести дані
- update – редагувати
- delete – видалити запис
- drop – видалити таблицю або запис

Основні запити формуються за допомогою ОРМ SQLAlchemy. Приклад запиту, який формується ОРМ:

```
select * from resume inner join employee on employee.id == resume.employee_id where employee_id = 1 order by updated_at desc
```

Запит до бази даних на пошук та повернення всіх резюме користувача з id 1 відсортованих у порядку найновішого за датою внесення змін.

3.3.4 Обсяг пам’яті

На даному етапі БД займає ~1 ГБ. З часом об’єм занятої пам’яті збільшиться до 7-10 ГБ. Великий розмір бази зумовлений необхідністю збереження великої моделей лінійної регресії для підбору. В подальшому планується вирішення проблеми збереження моделі та перенесення кроку тренування на етап передбачення. Таким чином вдасться зменшити об’єм пам’яті до 3-5 ГБ у кінцевому варіанті.

Висновки до розділу

На етапі проектування та розробки бази даних було визначено основні вимоги до БД з боку користувачів, виділено основні сутності предметної області (Користувач, Резюме, Компанія, Вакансія) та допоміжні (Мова програмування, Навичка). На основі отриманих результатів вивчення предметної області, було побудовано діаграму потоків даних без декомпозиції та ЕР-діаграму – модель

даних. В подальшому було виявлено можливі обмеження зв’язків та полів БД, описано набори відношень між сутностями. На основі цього модель потоків даних було нормалізовано до третьої нормальної форми, описано основні атрибути та зв’язки кожної таблиці БД. Було прийнято рішення про використання СУБД PostgreSQL, описані типові SQL-запити з поправкою на використання pgSQL та SQLAlcemy.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ РІШЕННЯ

4.1 Засоби реалізації

Мовою програмування було обрано Python версії 3.10, яка на даний момент часу є найновішою версією цієї мови програмування. Для розробки було використано інтегроване середовище розробки PyCharm Professional, доступ до якого було отримано за студентською ліцензією. Для реалізації API обрано веб-фреймворк FastAPI, що обумовлено його асинхронністю, легкістю підтримки та деплойменту. Тестування вихідного коду здійснюється за допомогою фреймворку PyTest. Алгоритми машинного навчання реалізовано за допомогою вбудованої бібліотеки ScikitLearn. Для зручної розробки та подальшого деплойменту створено докер імеджі баз даних PostgreSQL та Redis, які можна розвернути як на локальній машині так і на сервері. Фінальна версія коду задеплоєна та розгорнута на базі лінукс-сервері.

4.2 Опис вхідних даних

Вхідні данні до запитів надаються у форматі JSON у тілі POST та PATCH реквестів (запитів). Для кожного ендпоїнту передбачено свій шаблон вхідних даних, який можна подивитися в документації SWAGER або викликавши допоміжний ендпоїнт. Шаблони даних для створення та оновлення даних основних сутностей не відрізняються.

4.2.1 Вхідні данні для створення або оновлення резюме

Для створення та оновлення даних резюме у тілі запиту необхідно заповнити наступну форму:

{

"employee_id": ідентифікаційний код користувача

"programming_language_id": ідентифікаційний код мови програмування

"description": опис резюме, що має складатися як мінімум з 10 символів

“programming_tools”: список id навичок, які необхідні для вакансії, розташованих в порядку зменшення пріоритету.

}

Документація SWAGER для схеми (рис. 4.1):

Рисунок 4.1 – SWAGER документація для створення резюме

Приклад запиту для створення резюме:

```
curl -X POST --location "http://127.0.0.1:8000/company" \
   -H "accept: application/json" \
   -H "api-key: API_KEY" \
   -H "Content-Type: application/json" \
   -d "{
      "employee_id": 1,
      "programming_language_id": 1,
      "description": "Some description",
      "programming_tools": [1, 2, 3, 4, 5]
     }"
```

4.2.2 Вхідні данні для створення або оновлення вакансії

Для створення та оновлення даних вакансії у тілі запиту необхідно заповнити наступну форму:

{

"company_id": ідентифікаційний код компанії

"programming_language_id": ідентифікаційний код мови програмування

"description": опис резюме, що має складатися як мінімум з 10 символів

“programming_tools”: список id навичок, які необхідні для вакансії, розташованих в порядку зменшення пріоритету.

}

Документація SWAGER для схеми (рис. 4.2):

Рисунок 4.2 – SWAGER документація для створення вакансії

Приклад запиту для створення вакансії:

```
curl -X 'POST' \
 'http://127.0.0.1:8000/vacancy' \
 -H 'accept: application/json' \
 -H 'Content-Type: application/json' \
 -d '{
      "company_id": 1,
      "programming_language_id": 1,
      "description": "Resume description",
      "programming_tools": [1, 2, 3, 4]
 }'
```

4.2.3 Вхідні данні для створення або оновлення компанії

Для створення та оновлення даних компанії у тілі запиту необхідно заповнити наступну форму:

{

"**name**": назва компанії, яка має складатися як мінімум з 2х символів,

"**creation_date**": дата створення компанії (має бути в ISO-форматі),

"**number_of_employees**": число співробітників, має бути цілочисельним,

"**description**": опис компанії, що має складатися як мінімум з 10 символів

```
}
```

Опис схеми, який представлено на сторінці документації API (рис. 4.3):

Рисунок 4.3 – Документація SWAGER для створення компанії

Приклад запиту для створення компанії:

```
curl -X 'POST' \
  'http://127.0.0.1:8000/company' \
  -H 'accept: application/json' \
  -H 'api-key: API_KEY' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "DonNU",
  "creation_date": "2022-08-21T15:17:06.201Z",
  "number_of_employees": 650,
  "description": "Vasyl Stus\' Donetsk National University"
}'
```

4.2.4 Вхідні данні для створення або оновлення співробітника

Схема тіла запиту для створення та оновлення співробітнику має наступний вигляд:

```
{
"name": ім’я співробідника, яке має складатися як мінімум з 2х символів,
"second_name": по-батькові співробідника, має складатися як мінімум з 2х символів,
"surname": прізвище співробідника, яке має складатися як мінімум з 2х символів,
"birthdate": дата народження співробітника (має бути в ISO-форматі),
```

"**description**": опис досвіду співробітника, що має складатися як мінімум з 10 символів

}

Документація SWAGER для схеми (рис 4.4):

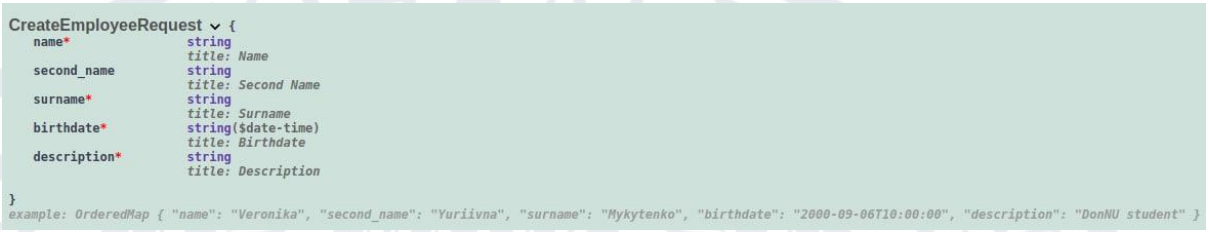

Рисунок 4.4 – Документація SWAGER для створення користувача

Приклад запиту на оновлення даних:

```
curl -X 'PATCH' \
 'http://127.0.0.1:8000/employee/1' \
 -H 'accept: application/json' \
 -H 'api-key: API_KEY' \
 -H 'Content-Type: application/json' \
 -d '{
 "name": "Veronika",
 "second_name": "Yuriivna",
 "surname": "Mykytenko",
 "birthdate": "2000-09-06T10:00:00",
 "description": "DonNU student"
}'
```

4.3 Опис класів та їх методів

Дотримуючись принципів інкапсуляції, зокрема правилу розділення інтерфейсу та реалізації та основним концептам SOLID, весь код розділено на 7 основних модулів:

1. core – файли, що зберігають в собі константи та класи конфігурацій
2. crud – класи для виконання основних операцій з сутностями бази даних

3. db – класи та методи що контролюють підключення до бази даних та створення сесій
4. models – ORM-класи для створення та маніпуляцій із сутностями бази даних
5. routers – набір методів для обробки HTTP-запитів
6. schema – pydantic класи-моделі для валідації HTTP запитів
7. utils – набір допоміжних функцій- та класів-утиліт.

У корні репозиторію також знаходяться файл конфігурацій мідлварів, методи для валідації ключа, що надається в запиті з хедером api-key, кастомні класи помилок та основний контроллер додатку.

4.3.1 CRUD класи

CRUD-класи – це класи, що інкапсулюють в собі методи виконання основних операцій із сутностями бази даних – Create Read Update Delete. В таких класах містяться функції, що всередині маніпулюють інстансами ORM-моделей, переданих у конструктор класу (згідно дизайн-патерну Dependency Injection об’єкт ORM-класу не має ініціалізуватися у конструкторі іншого класу, а має бути переданим як аргумент функції __init__).

CompanyCRUD

Клас для маніпулювання даними компаній має назву CompanyCRUD. Він розташований за адресою ~/it_worksearch_app/crud/company_crud.py.

Клас містить в собі наступні методи:

- create_company – метод для створення нової компанії в базі даних. Дані отримуються з тіла запиту, надісланого на ендпоїнт та переданого до методу об’єктом класу CreateCompanyRequest. Повертає об’єкт класу CompanyResponse, що представляє собою рядок таблиці Company.
- read_company_by_id – метод що зчитує із бази даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Повертає об’єкт класу CompanyResponse, що представляє собою рядок таблиці Company.

- read_all_companies – метод що зчитує всі записи таблиці Company та повертає об’єкт типу list, що містить в собі об’єкти класу CompanyResponse
- update_company - метод що оновлює в базі даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Дані отримуються з тіла запиту, надісланого на ендпоїнт та переданого до методу об’єктом класу UpdateCompanyRequest. Повертає об’єкт класу CompanyResponse, що представляє собою рядок таблиці Company.
- delete_company_by_id – метод що видаляє із бази даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Повертає строку з результатом видалення.

EmployeeCRUD

Клас для маніпулювання даними робітників має назву EmployeeCRUD. Він розташований за адресою ~/it_worksearch_app/crud/employee_crud.py.

Клас містить в собі наступні методи:

- create_employee – метод для створення нового користувача в базі даних. Дані отримуються з тіла запиту, надісланого на ендпоїнт та переданого до методу об’єктом класу CreateEmployeeRequest. Повертає об’єкт класу EmployeeResponse, що представляє собою рядок таблиці Employee.
- read_employee_by_id – метод що зчитує із бази даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Повертає об’єкт класу EmployeeResponse, що представляє собою рядок таблиці Employee.
- read_all_employees – метод що зчитує всі записи таблиці Company та повертає об’єкт типу list, що містить в собі об’єкти класу EmployeeResponse
- update_employee - метод що оновлює в базі даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у

метод як аргумент типу даних int. Дані отримуються з тіла запиту, надісланого на ендпоїнт та переданого до методу об’єктом класу UpdateEmployeeRequest. Повертає об’єкт класу EmployeeResponse, що представляє собою рядок таблиці Company.

- delete_employee_by_id – метод що видаляє із бази даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Повертає строку з результатом видалення.

ResumeCRUD

Клас для маніпулювання даними резюме має назву ResumeCRUD. Він розташований за адресою ~/it_worksearch_app/crud/resume_crud.py.

Клас містить в собі наступні методи:

- create_resume – метод для створення нового резюме в існуючого користувача в базі даних. Дані отримуються з тіла запиту, надісланого на ендпоїнт та переданого до методу об’єктом класу CreateResumeRequest. Повертає об’єкт класу ResumeResponse, що представляє собою рядок таблиці Resume.
- read_resume_by_id – метод що зчитує із бази даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Повертає об’єкт класу ResumeResponse, що представляє собою рядок таблиці Resume.
- read_all_resumes – метод що зчитує всі записи таблиці Resume та повертає об’єкт типу list, що містить в собі об’єкти класу ResumeResponse
- update_resume - метод що оновлює в базі даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Дані отримуються з тіла запиту, надісланого на ендпоїнт та переданого до методу об’єктом класу UpdateResumeRequest. Повертає об’єкт класу ResumeResponse, що представляє собою рядок таблиці Resume.

- delete_resume_by_id – метод що видаляє із бази даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Повертає строку з результатом видалення.

VacancyCRUD

Клас для маніпулювання даними компаній має назву VacancyCRUD. Він розташований за адресою ~/it_worksearch_app/crud/vacancy_crud.py.

Клас містить в собі наступні методи:

- create_vacancy – метод для створення нової вакасії вже існуючої компанії в базі даних. Дані отримуються з тіла запиту, надісланого на ендпоїнт та переданого до методу об’єктом класу CreateVacancyRequest. Повертає об’єкт класу VacancyResponse, що представляє собою рядок таблиці Vacancy.
- read_vacancy_by_id – метод що зчитує із бази даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Повертає об’єкт класу VacancyResponse, що представляє собою рядок таблиці Vacancy.
- read_all_vacancies – метод що зчитує всі записи таблиці Vacancy та повертає об’єкт типу list, що містить в собі об’єкти класу VacancyResponse
- update_vacancy - метод що оновлює в базі даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Дані отримуються з тіла запиту, надісланого на ендпоїнт та переданого до методу об’єктом класу UpdateVacancyRequest. Повертає об’єкт класу VacancyResponse, що представляє собою рядок таблиці Vacancy.
- delete_vacancy_by_id – метод що видаляє із бази даних один запис з заданим id. Id отримується із параметрів url викликаного ендпоїнту та передається у метод як аргумент типу даних int. Повертає строку з результатом видалення.

4.3.2 Session класи

Для роботи із базою даних PostgreSQL в асинхронному режимі необхідно використати відповідний тип конектора, який дозволяв би виконувати CRUD-операції асинхронно, уникнувши блокувань потоків даних. Для цього створено допоміжний клас SessionFactory, що реалізує паттерн програмування «Фабрика». Цей клас дозволятиме створювати незалежну сессію для кожного запиту, що буде відповідати за транзакції бази даних.

Кожна сессія, отримана в результаті ітерування фабрикою буде мати тип даних AsyncSession.

4.3.3 Router клас

Клас, що відповідає за призначення конкретної функції-обробника для кожного HTTP-запиту створюється за допомогою вбудованого класу фреймворку FastAPI. Кожна функція-обробник додається до класу за допомогою декоратора, що визначає тип запиту, url який буде обробляти функція, типи кодів відповідей, та модель відповіді.

Функції-обробники мають бути асинхроними, та одним із аргументів приймати сессію типу AsyncSession. Кожна функція також має містити базовий обробник помилок, що дозволить в разі визначених типів помилки (помилки валідації, таймаут помилки і т.д) поверталися з відповідним кодом та повідомленням.

Приклад декоратору для додання функції-обробника GET-запиту на створення нової компанії:

```
@router.post(    # визначення типу запиту, що приймається для обробки
  "/company",  # url з якого надходитиме запит
  responses={  # типи кодів відповідей серверу та їх описи
    201: {"description": "Company with specified data was created"},
    403: {"model": Error, "description": "Invalid api-key header value"},
    406: {"model": Error, "description": "Passed data format invalid"},
    422: {"model": Error, "description": "Passed data format invalid"},
```

```
    },
    response_model=CompanyResponse,  # модель відповіді
    status_code=status.HTTP_201_CREATED  # код статусу за замовченням
)
```

Приклад асинхроної функції-обробника для створення нової компанії:

```
async def registrate_company(request: CreateCompanyRequest, session: AsyncSession = Depends(get_session)) -> CompanyResponse:
    try:
        logger.info(f"creating new company: {request=}")
        response = await CompanyCRUD(db_session=session).create_company(request=request)
        return response
    except ValidationError as e:
        raise HTTPException(status_code=status.HTTP_406_NOT_ACCEPTABLE, detail=e)
    except TimeoutError:
        raise
    except HTTPException:
        raise
    except BaseITWorkSearchException:
        raise
    except Exception as e:
        log_traceback(extra_message=str(e))
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

4.4 Опис реалізації алгоритму машинного навчання

Фактично, реалізація алгоритму машинного навчання складається з трьох частин – підготовка даних для моделей, створення моделі лінійної регресії та тренування моделі.

Для підготовки даних необхідно виконати select запит про вакансію або резюме в базі даних для отримання інформації про programming_tools з їх пріоритетами для вакансії. Далі, сумуються всі коефіцієнти навичок, щоб отримати коефіцієнт сутності.

Наступним кроком стає створення моделі лінійної регресії за допомогою інструментів бібліотеки ScikitLearn. Створену модель для подальшого використання необхідно натренувати на готових даних, що стає третім кроком. Натреновану модель кодуємо інструментами base64 та записуємо в допоміжну таблицю з ключом id резюме або вакансії та типом сутності.

4.4.1 Створення моделі лінійної регресії

Для створення моделі лінійної регресії використано вбудований клас LinearRegression з модулю linear_model бібліотеки sklearn.

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

Аргументами конструктору __init__ можуть виступити:

- fit_intercept – логічний (True за замовчуванням) параметр, який вирішує, обчислювати відрізок b_0 (True) або розглядати його як нуль (False).
- normalize – логічний (False за замовчуванням) параметр, який вирішує, нормалізувати вхідні змінні (True) чи ні (False).
- copy_X – логічний (True за промовчанням) параметр, який вирішує, копіювати (True) або перезаписувати вхідні змінні (False).

- n_jobs – ціле або None (за умовчанням), що становить кількість процесів, задіяних у паралельних обчисленнях. None означає відсутність процесів, при -1 використовуються всі доступні процесори.

Фактично для роботи з лінійною регресією в нашому випадку підходять параметри за замовченням. Змінна model буде використовуватися в подальшому для роботи з регресією.

4.4.2 Тренування моделі

Для тренування моделі для початку треба підготувати ndarray об’єкти x та y, де x – масив із чотирьома величинами коефіцієнту (100% коефіцієнту, 75% коефіцієнту, 50% коефіцієнту та 25% коефіцієнту) та y – масив констант з відповідністю до відсотків – 100, 75, 50, 25.

```
coefficients_array = [coefficient, coefficient*75/100, cofficient*50/100, cofficient*25/100]
constants_array = [100, 75, 50, 25]

x = np.array(coefficients_array).reshape(-1, 1)
y = np.array(constants_array)
```

Наступним кроком використаємо модель, яку було створено в попередньому кроці та тренуємо на створених даних для отримання коефіцієнту детермінації.

```
model.fit(x, y)
r_sq = model.score(x, y)
logger.info('coefficient of determination:', r_sq)
```

Останнім кроком серіалізуємо модель у base64 кодування та записуємо отриману байт-стрічку у допоміжну таблицю бази даних.

4.4.3 Використання моделі для передбачення проценту сумісності вакансії і резюме

Для подальшого використання моделі для підбору найліпшого варіанту протилежної сутності необхідно виконати select запит до бази даних, отримати байт стрічку, розкодувати та десеріалізувати модель лінійної регресії та використати метод predict на вибірці сутностей, що задовольняють умову збіжності мов програмування. Розберемо повний workflow алгоритму підбору найкращого резюме до вакансії на прикладі.

Припустимо, маємо вакансію для мови програмування python та необхідними навичками [“fastAPI”, “Redis”, “SciKitLearn”, “ci-cd knowledge”, “pandas”]. У таблиці коефіцієнтів роботодавець визначив наступні ціннісні коефіцієнти для навичок:

- “fastAPI” – 14.56324
- “Redis” – 12.23546
- “SciKitLearn” – 9.34588
- “ci-cd knowledge” – 5.43274
- “pandas” – 2.65185

Таким чином коефіцієнт вакансії становить 44.22917.

За формулою отримаємо наступні масиви даних для x та y:

```
x = [44.22917, 33.1718775, 22.114585, 11.0572925]
y = [100, 75, 50, 25]
```

Далі select запитом отримаємо з бази даних всі резюме, які мають мову програмування python та хоча б одну з навичок. Маємо наступні варіації:

1. [“fastAPI”, “pandas”]
2. [“SciKitLearn”, “ci-cd knowledge”, “pandas”]
3. [“fastAPI”, “Redis”, “pandas”]
4. [“fastAPI”, “SciKitLearn”, “ci-cd knowledge”]

Таким чином отримуємо вибірку коефіцієнтів резюме:

```
[17.21509, 17.43047, 29.45055, 29.34186]
```

На отриманій натренованій моделі, використовуємо метод predict та отримаємо наступні процентні співвідношення сумісності резюме та вакансії:

1. 38.92248034 %
2. 39.40944404 %
3. 66.5862597 %
4. 66.3405169 %

Відсортувавши за спаданням отримаємо результати, що найбільш вдалим кандидатом на вакансію буде співробітник з id 3 – 66,59% збіжності.

4.5 Вимоги до ПЗ та АЗ

Програма повинна працювати на операційній системі Linux дистрибутивів Ubuntu, Mint, Debian.

До складу технічних засобів повинен входити IBM-сумісний персональний комп'ютер (ПЕОМ), що включає в себе:

- процесор, не менше intel core i3 7^{th} gen;
- оперативну пам'ять, об'ємом не менше 4 Gb;
- маніпулятор типу "миша";
- Пристрій для введення – клавіатура.

Також для роботи програми повинні бути встановлені сервери Redis, PostgeSQL, IDE PyCharm.

Для запуску серверу необхідно мати доступ до інтернету і високороздільну здатність системи.

4.6 Інструкції для користувача

4.6.1 Авторизація для використання ендпоїнтів

Для авторизації запиту до ендпоїнтів, що надсилається через утиліту консолі curl або через клієнт http-запитів, на кшталт Postman, потрібно вказати хедер запиту “api_key” та передати в нього декодований ключ доступу до API (рис. 4.5).

Рисунок 4.5 – приклад авторизації

Для авторизації для тестування ендпоїнтів на сторінці SWAGER необхідно у правому верхньому кутку натиснути на кнопку “Authorize” та в спливаючому вікні додати декодований ключ авторизації та натиснути на кнопку “Authorize” (рис 4.6). Надалі всі запити, які будуть відправлені зі сторінки SWAGER будуть проходити валідацію ключа (рис. 4.7).

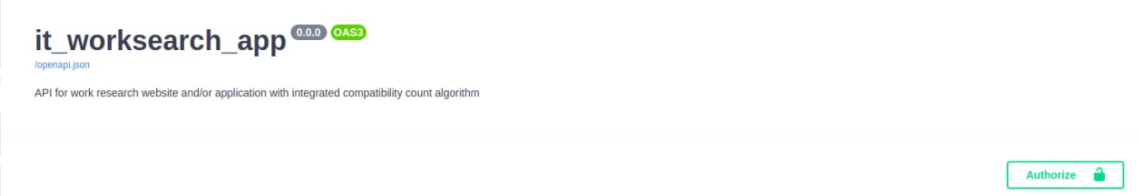

Рисунок 4.6 – авторизація на сторінці SWAGER документації

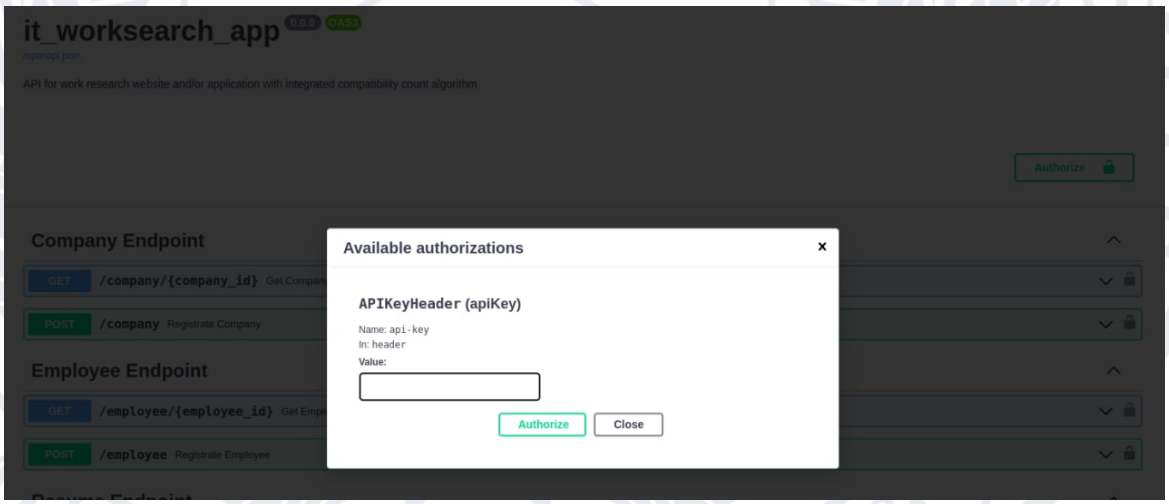

Рисунок 4.7 – авторизація на сторінці SWAGER документації

4.6.2 Опис ендпоїнтів та вхідних даних для їх використання

Всього в додатку є п’ять типів ендпоїнтів – 4 для основних сутностей Компанії, Працівники, Резюме та Вакансії та ендпоїнти для використання алгоритмів машинного навчання.

Company ендпоїнти

Для сутності Компанія реалізовано чотири ендпоїнти які відповідають за основні операції CRUD (рис. 4.8).

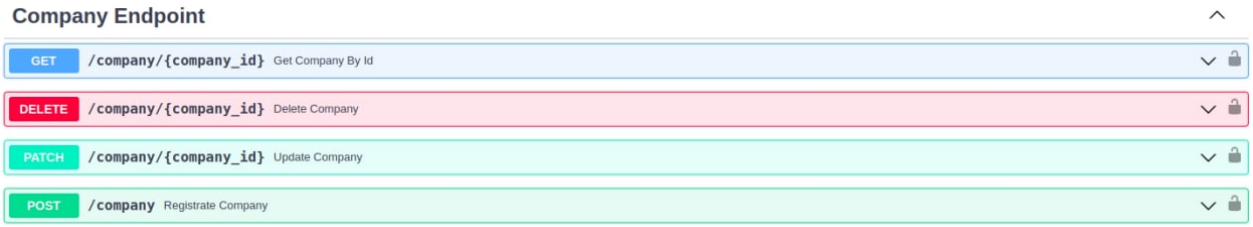

Рисунок 4.8 – ендпоїнти сутності Компанія

Посилання /company/company_id може отримувати запити трьох типів:

GET запити з хедером api_key, параметром company_id – id запису бази даних яку необхідно отримати. Запит повертає запис з таблиці з id=company_id.

DELETE запити з хедером api_key, параметром company_id – id запису бази даних яку необхідно видалити. Запит видаляє запис з таблиці де id=company_id.

PATCH запити з хедером api_key, параметром company_id та тілом, в якому передаються дані формату json відповідно до схеми UpdateCompanyRequest. Запит оновлює запис в базі даних з id=company_id та повертає оновлені дані.

Для створення запису компанії необхідно надіслати запит на ендпоїнт /company типу POST хедером api_key та тілом, в якому передаються дані формату json відповідно до схеми CreateCompanyRequest. Запит створює новий запис в таблиці та повертає новий рядок із БД.

Employee ендпоїнти

Для сутності Користувач реалізовано чотири ендпоїнти які відповідають за основні операції CRUD (рис. 4.9).

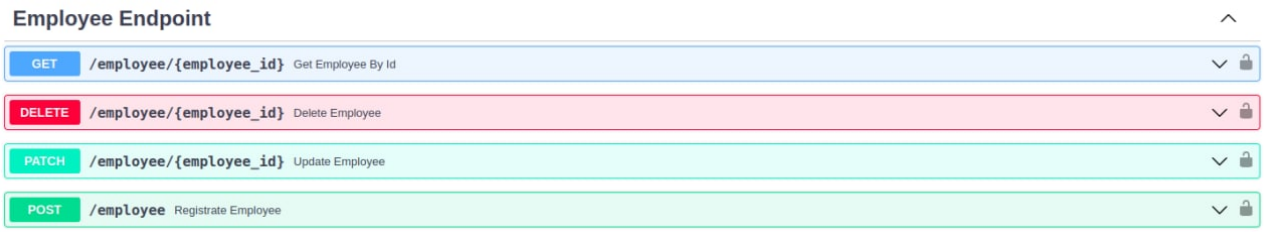

Рисунок 4.9 – ендпоїнти сутності Користувач

Посилання /employee/ employee_id може отримувати запити трьох типів:

GET запити з хедером api_key, параметром employee_id – id запису бази даних яку необхідно отримати. Запит повертає запис з таблиці з id= employee_id.

DELETE запити з хедером api_key, параметром employee_id – id запису бази даних яку необхідно видалити. Запит видаляє запис з таблиці де id=employee_id.

PATCH запити з хедером api_key, параметром employee_id та тілом, в якому передаються дані формату json відповідно до схеми UpdateEmployeeRequest. Запит оновлює запис в базі даних з id=employee_id та повертає оновлені дані.

Для створення нового користувача необхідно надіслати запит на ендпоїнт /employee типу POST хедером api_key та тілом, в якому передаються дані формату json відповідно до схеми CreateEmployeeRequest. Запит створює новий запис в таблиці та повертає новий рядок із БД.

Resume ендпоїнти

Для сутності Резюме реалізовано чотири ендпоїнти які відповідають за основні операції CRUD (рис. 4.10).

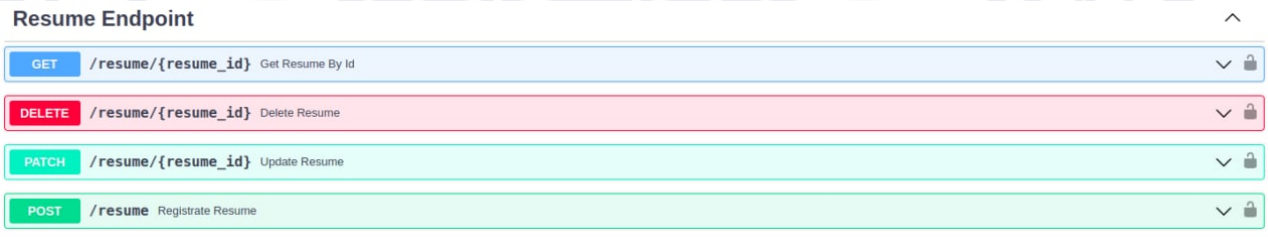

Рисунок 4.10 – ендпоїнти сутності Резюме

Посилання /resume/resume_id може отримувати запити трьох типів:

GET запити з хедером api_key, параметром resume_id – id запису бази даних яку необхідно отримати. Запит повертає запис з таблиці з id=resume_id.

DELETE запити з хедером api_key, параметром resume_id – id запису бази даних яку необхідно видалити. Запит видаляє запис з таблиці де id=resume_id.

PATCH запити з хедером api_key, параметром company_id та тілом, в якому передаються дані формату json відповідно до схеми UpdateResumeRequest. Запит оновлює запис в базі даних з id= resume_id та повертає оновлені дані.

Для створення запису резюме необхідно надіслати запит на ендпоїнт /resume типу POST хедером api_key та тілом, в якому передаються дані формату json відповідно до схеми CreateResumeRequest. Запит створює новий запис в таблиці та повертає новий рядок із БД.

Vacancy ендпоїнти

Для сутності Вакансія реалізовано чотири ендпоїнти які відповідають за основні операції CRUD (рис. 4.11).

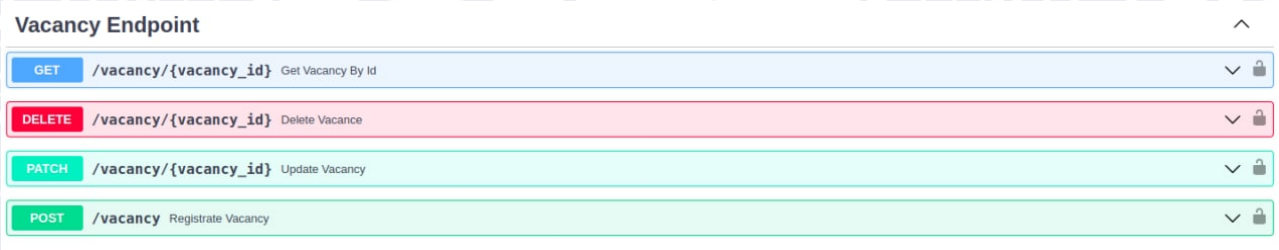

Рисунок 4.11 – ендпоїнти сутності Вакансія

Посилання /vacancy/vacancy_id може отримувати запити трьох типів:

GET запити з хедером api_key, параметром vacancy_id – id запису бази даних яку необхідно отримати. Запит повертає запис з таблиці з id=vacancy_id.

DELETE запити з хедером api_key, параметром vacancy_id – id запису бази даних яку необхідно видалити. Запит видаляє запис з таблиці де id=vacancy_id.

PATCH запити з хедером api_key, параметром vacancy_id та тілом, в якому передаються дані формату json відповідно до схеми UpdateVacancyRequest. Запит оновлює запис в базі даних з id=vacancy_id та повертає оновлені дані.

Для створення запису вакансії необхідно надіслати запит на ендпоїнт /vacancy типу POST хедером api_key та тілом, в якому передаються дані формату

json відповідно до схеми CreateVacancyRequest. Запит створює новий запис в таблиці та повертає новий рядок із БД.

Ендпоїнти для підбору вакансій та резюме

Ендпоїнти, які відповідають за підбір резюме та вакансій мають посилання, яке починається з /select (рис. 4.12).

Рисунок 4.12 – ендпоїнти для підбору резюме та вакансій

Для підбору найліпшої вакансії для резюме необхідно надіслати GET запит на ендпоїнт /select/for_resume/resume_id з параметром resume_id – id запису в таблиці резюме та хедером api_key.

Для підбору резюме, що найбільш відповідає критеріям вакансії необхідно надіслати GET запит на ендпоїнт /select/for_vacancy/vacancy_id з параметром vacancy_id – id запису в таблиці вакансія та хедером api_key.

4.6.3 SWAGGER документація

Для перегляду документації по існуючим ендпоїнтам, тестування надсилання повідомлень та ознайомлення з моделями даних вхідних запитів необхідно надіслати GET запит на ендпоїнт /docs. Отримана інформація буде представлена у форматі json. Також можна перейти за тим самим url в браузері можна відкрити веб версію з фронтендом, реалізованим завдяки бібліотеці SWAGGER (рис. 4.13).

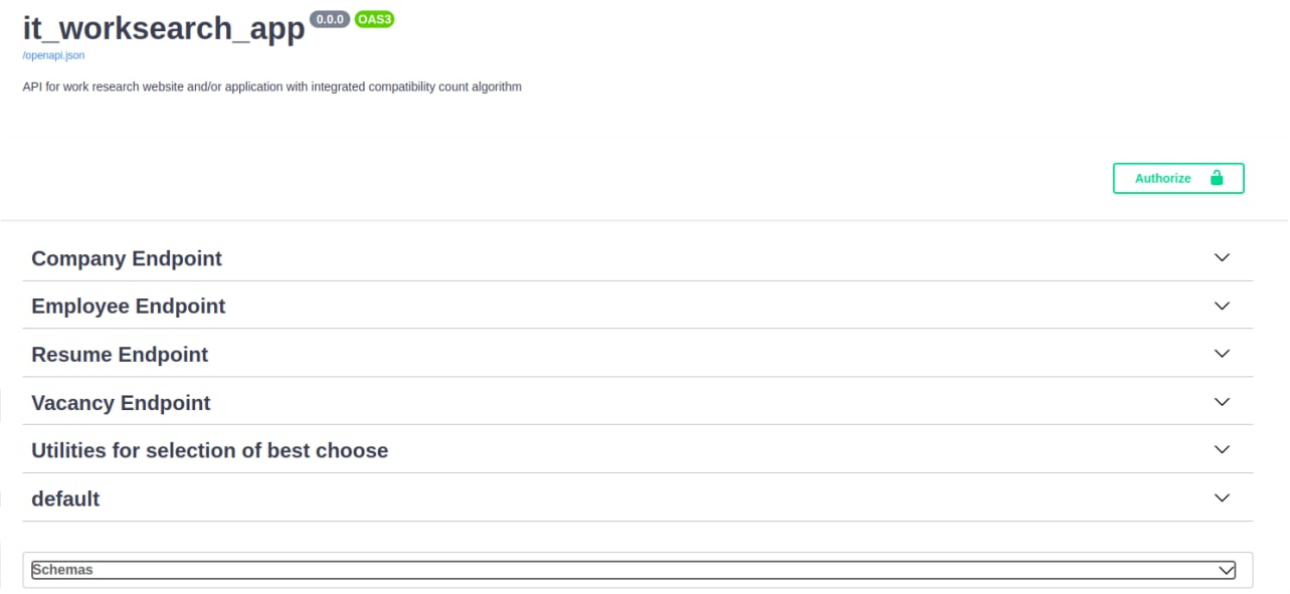

Рисунок 4.13 – сторінка SWAGGER документації

4.7 Тестування розробленого ПЗ

Для розробленого програмного забезпечення було проведено функціональне (unit- та інтеграційне тестування) та нефункціональне тестування (performance та end-to-end тестування). За вимогами до ПЗ необхідно як мінімум 80% покриття коду автоматичними тестами. Обов’язковим є стрес-тестування для алгоритмів машинного навчання.

4.7.1 Unit-тести

Цей метод тестування програмного забезпечення полягає в окремому тестуванні кожного модуля коду програми. Для цього за допомогою бібліотеки pytest було створено функції, які приймають на вхід зразки даних та перевіряють окремі ендпоїнти на відповідність вихідних даних. Особливість юніт-тестування в відсутності зв’язку із зовнішніми сервісами, такими як Redis, PostgreSQL та інші. Тест-coverage приведено в таблиці 4.1.

Таблиця 4.1 - Покриття Unit-тестами.

Ендпоїнт	Статус модульного тестування
GET /company/{company_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для отримання даних. Покрито позитивні та негативні тест-кейси.
DELETE /company/{company_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для видалення даних. Покрито позитивні та негативні тест-кейси.
PATCH /company/{company_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для оновлення даних. Покрито позитивні та негативні тест-кейси.
POST /company	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для додавання даних. Покрито позитивні та негативні тест-кейси.
GET /employee/{employee_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для отримання даних. Покрито позитивні та негативні тест-кейси.
DELETE /employee/{employee_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для видалення даних. Покрито позитивні та негативні тест-кейси.

Продовження таблиці 4.1

PATCH /employee/{employee_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для оновлення даних. Покрито позитивні та негативні тест-кейси.
POST /employee	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для додавання даних. Покрито позитивні та негативні тест-кейси.
GET /resume/{resume_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для отримання даних. Покрито позитивні та негативні тест-кейси.
DELETE /resume/{resume_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для видалення даних. Покрито позитивні та негативні тест-кейси.
PATCH /resume/{resume_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для оновлення даних. Покрито позитивні та негативні тест-кейси.
POST /resume	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для додавання даних. Покрито позитивні та негативні тест-кейси.
GET /vacancy/{vacancy_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для отримання даних. Покрито позитивні та негативні тест-кейси.

Продовження таблиці 4.1

DELETE /vacancy/{vacancy_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для видалення даних. Покрито позитивні та негативні тест-кейси.
PATCH /vacancy/{vacancy_id}	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для оновлення даних. Покрито позитивні та негативні тест-кейси.
POST /vacancy	Тестування проведено частково з mock-ом на PGSQL-клієнті та перевіркою SQL-квері, яка використовується для додавання даних. Покрито позитивні та негативні тест-кейси.
GET /select/for_resume/{resume_id}	Тестування проведено повністю, покриті позитивні та негативні тест-кейси, покрито як повне проходження алгоритму, так і його частини.
GET /select/for_vacancy/{vacancy_id}	Тестування проведено повністю, покриті позитивні та негативні тест-кейси, покрито як повне проходження алгоритму, так і його частини.

4.7.2 Інтейграційне тестування

Цей тип тестування відбувається для перевірки інтеграції зовнішніх компонентів до основного функціоналу програми. Тест-coverage приведено в таблиці 4.2

Таблиця 4.2 - Покриття інтеграційними тестами.

Сервіси	Статус тестування
Redis	Створено тесткейс для перевірки конекту до Redis-серверу. Для тестування створено докер контейнер, який підіймається до тестування, в новий інстанс Redis додаються нові ключі api_key, та в самому тесті перевіряється можливість оцінки відповідності переданого ключа до того, що зберігається у БД. Покрито позитивний та негативний тесткейс, який перевіряє підіймання помилки з кодом 403.
PostgreSQL	Проведено тестування кожного ендпоїнту без mock-у клієнту БД. Перед проведенням тестування підіймається докер контейнер з чистою базою даних, в яку за допомогою запитів додаються, читаються, оновлюються та видаляються дані.

4.7.3 Performance тестування

Тестування продуктивності було проведено для двох ендпоїнтів, що відповідають за підбір найкращих кандидатів/вакансій. Для тесту було використано ноутбук з 8-ядерним процесором Intel Core I7 9th Gen, 16 Gb RAM. Була зроблена вибірка з 10000 резюме, які можуть підійти для однієї вакансії. Повний час підбору першого тесту – 8 хвилини 34 секунди. Було також помічено 5 TimeoutError помилок від бази даних, через надмірне навантаження PostgreSQL серверу.

Після аналізу результатів було внесено зміни до таблиць бази даних – було додано індексацію записів, що суттєво зменшило час на обробку даних. Повний час підбору другого тесту – 3 хвилини 56 секунд. Помилок не було.

4.7.4 End-to-end тестування

Такий тип тестування передбачає максимально повне тестування всього workflow. Для цього було виконані наступні кроки:

1. Створення нового користувача
2. Створення нового резюме
3. Створення нової компанії
4. Створення чотирьох новий вакансій для мови python з різними навичками
5. Підбір найліпшої вакансії для створеного резюме.

Тестування було пройдено успішно – кожен із кроків завершився без помилок, з очікуваним результатом.

Висновки до розділу

В ході роботи над розділом було описано типові формати вхідних даних, модулі розробленого додатку. Окремо, більш детально було розібрано архітектуру чотирьох CRUD-класів, які використовуються як проміжний шар для роботи з БД. Опрацьовано архітектуру Session- та Router-класів, детально описано їх функції, надано приклади лістингів реалізації обробників запитів. Окремим пунктом під час розробки розділу було виділено опис реалізації алгоритмів машинного навчання з детальним розбором на основі наведеного прикладу. Додатково було розроблено інструкції для користувача, які містять в собі опис процесу аутентифікації. Існуючі ендпоїнти для сутностей Компанія, Вакансія, Користувач та Резюме також ендпоїнти для підбору найбільш сумісних вакансій та резюме. Описано документацію, побудовану на основі фреймворку SWAGGER. Розроблено автоматичні юніт та інтеграційні тести. Наведено опис результатів Perfomance та End-to-End тестування.

ВИСНОВКИ

В ході виконання роботи було виконано всі поставлені задачі:

- Було розроблено сервісну частину веб-API та реалізовано 18 ендпоїнтів для виконання CRUD операцій над даними компаній, користувачів, резюме та вакансій а також використання вбудованої системи підбору.
- Створено базу даних яка містить в собі 6 таблиць відображаючи основні сутності, які використовуються в додатку – Вакансія, Резюме, Користувач, Компанія, Навичка та Мова програмування. У таблицях проведено індексацію даних для вирішення performance проблем під час читання великої кількості рядків.
- Реалізовано механізм аутентифікації запитів до серверу за допомогою бази даних Redis, в якій зберігаються ключі до API, дозволенні для використання у хедері api-key запиту.
- Створено сторінку документації на базі сервісу SWAGGER з можливістю тестування ендпоїнтів, описом моделей вхідних та вихідних даних.
- Unit- та інтеграційними тестами покрито 86,37% функціоналу, проведено end-to-end та performance тестування, результати яких опубліковано в роботі.
- Розроблено скрипти для створення докер-файлів, cicd налаштування деплойменту на сервер та файли автоматичного тестування покриття коду.
- Імплементовано алгоритми машинного навчання на основі моделі лінійної регресії та передбачення та підбору найбільш сумісних вакансій та резюме.

Результати роботи в подальшому можуть бути розширені за допомогою додання нових ендпоїнтів для маніпуляції даними (наприклад, для отримання списку всіх резюме користувача) або реалізації додаткових алгоритмів

машинного навчання (наприклад, передбачення заробітної плати по резюме за вказаними навичками).

Фінальним результатом роботи є сервер Application Programming Interface, який може маніпулювати даними користувачів, компаній, резюме та вакансій, підбирати найбільш сумісні резюме та вакансії. Користувач отримує як інструментарій для розробки подальших продуктів (веб-сайту з фронтендом, десктоп додатку або мобільної версії для Android чи IOS), створення інших мікросервісів, які б використовували ендпоїнти створеного API та веб-сторінку з документацією та можливістю використання ендпоїнтів додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What Is An Api? URL: https://aws.amazon.com/what-is/api/ (дата звертання: 15.05.2022).

2. REST APIs. URL: https://developers.planet.com/docs/planetschool/rest-apis/?gclid=Cj0KCQjw166aBhDEARIsAMEyZh7TBIqJGClsO-Ux4ZfpRlYYHlj4N203JFrvAgCZBOHAYfXfqj_kfbMaAoDdEALw_wcB (дата звертання: 15.05.2022).

3. What is REST. URL: https://restfulapi.net/ (дата звертання: 15.05.2022).

4. SOAP API. URL: https://stoplight.io/api-types/soap-api (дата звертання: 15.05.2022).

5. SOAP vs REST. What's the Difference? URL: https://smartbear.com/blog/soap-vs-rest-whats-the-difference/ (дата звертання: 15.05.2022).

6. 14 Most Important Python Features and How to Use them. URL: https://www.simplilearn.com/python-features-article (дата звертання: 18.05.2022).

7. Asynchronous vs. Synchronous Programming: Key Similarities and Differences. URL: https://www.mendix.com/blog/asynchronous-vs-synchronous-programming/ (дата звертання: 18.05.2022).

8. Python & APIs: A Winning Combo for Reading Public Data. URL: https://realpython.com/python-api/#requests-and-apis-a-match-made-in-heaven (дата звертання: 18.05.2022).

9. Django Web Framework (Python). URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django (дата звертання: 18.05.2022).

10. FastAPI. URL: https://fastapi.tiangolo.com/ (дата звертання: 18.05.2022).

11. Чому я обираю FastAPI: основні можливості та переваги фреймворку. URL: https://dou.ua/forums/topic/37547/ (дата звертання: 18.05.2022).

12. How Python can be used in Machine Learning? URL: https://www.edoxitraining.com/studyhub-detail/how-python-can-be-used-in-machine-learning (дата звертання: 12.06.2022).

13. A Simple Guide to Scikit-Learn — Building a Machine Learning Model in Python. URL: https://towardsdatascience.com/a-beginners-guide-to-text-classification-with-scikit-learn-632357e16f3a (дата звертання: 29.05.2022).

14. SQLAlchemy. URL: https://github.com/sqlalchemy/sqlalchemy (дата звертання: 29.05.2022).

15. alembic 1.8.1. URL: https://pypi.org/project/alembic/ (дата звертання: 29.05.2022).

16. Redis. URL: https://hub.docker.com/_/redis (дата звертання: 29.05.2022).

17. Uvicorn. URL: https://www.uvicorn.org (дата звертання: 29.05.2022).

18. HTTP. URL: https://uk.wikipedia.org/wiki/HTTP (дата звертання: 29.05.2022).

19. HTTPS. URL: https://uk.wikipedia.org/wiki/HTTPS (дата звертання: 29.05.2022).

20. What is a Proxy Server? How They Work + List of Security Risks. URL: https://www.upguard.com/blog/proxy-server (дата звертання: 29.05.2022).

21. Linear regression. URL: https://www.ibm.com/topics/linear-regression (дата звертання: 17.06.2022).

22. Linear Regression in Python. URL: https://realpython.com/linear-regression-in-python/#linear-regression (дата звертання: 17.06.2022).

23. Січко Т.В., Юрчук Б.О. Розробка веб-додатку туристичної фірми. Вісник Хмельницького національного університету. Технічні науки. №4. 2018. С. 166-172.

24. Raschka S., Mirjalili V. Python Machine Learning – Second Edition. Birmingham: Packt Publishing, 2017. 622 с.

25. Chan J., Chung R., Huang J. Python API Development Fundamentals. Birmingham: Packt Publishing, 2019. 372 с.

Микитенко Вероніка Юріївна

Факультет інформаційних та прикладних технологій

122 «Комп'ютерні науки»

«Комп'ютерні технології обробки даних»

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (магістерська) робота на тему: «Розробка веб-додатку пошуку роботи з інтелектуальною системою підбору» є написана мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувалась іншими особами, а також дані та інформація не отримувались у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

_____ _____

(дата) (підпис здобувача освіти)